12. The TMS34010 Instruction Set

This section contains the TMS34010 instruction set (in alphabetical order). Related subjects, such as addressing modes, are presented first.

12.1 Symbols and Abbreviations

The symbols and abbreviations in Table 12-1 are used in the addressing modes discussion, the instruction set summary, and in the individual instruction descriptions.

Table 12-1. TMS34010 Instruction Set Symbol and Abbreviation Definitions

Symbol	Definition	Symbol	Definition
Register File A	Registers A0–A14, including SP	Register File B	Registers B0–B14, including SP
Rs	Source register	Rd	Destination register
Rs**X**	X half of source register	Rs**Y**	Y half of source register
Rd**X**	X half of destination register	Rd**Y**	Y half of destination register
An	Register *n* in register file A	Bn	Register *n* in register file B
PC	Program counter	PC′	PC prime. Specifies the PC of the next instruction (PC + instruction length)
ST	Status register	N	Status sign bit
C	Status carry bit	Z	Status zero bit
V	Status overflow bit	IE	Global interrupt enable bit
SP	Stack pointer	TOS	Top of stack
SAddress	Source address	DAddress	Destination address
MSW	Most significant word	LSW	Least significant word
LSB	Least significant bit	MSB	Most significant bit
>	Hexadecimal number	K	5-bit constant
IW	16-bit immediate value	IL	32-bit immediate value
W	16-bit immediate value	L	32-bit immediate value
F	Field select. **F=0** selects FS0, FE0 in the status register, **F=1** selects FS1, FE1	R	Register file select. Indicates which register file (A or B) the operand registers are in. **R=0** specifies register file A, **R=1** specifies register file B
( )	In **instruction syntax**, contents of. For example, (Rd) specifies the contents of the destination register	:	Concatenation. For example, Rd:Rd + 1 means the concatenation of one register and the next into a 64-bit value, as in A0:A1
→	Becomes the contents of	~	1's complement
| |	Absolute value	[ ]	Optional parameter
*	Indirect addressing	@	Absolute addressing
<text>	In **instruction syntax**, indicates a "fill in the blank" – substitute an actual value, address, or register for the text enclosed in the angle brackets. For example, substitute an actual source register for *<Rs>*; substitute an actual destination address for *<DAddress>*.		

12.2 Addressing Modes

The TMS34010 supports a variety of addressing modes. Most instructions use only one addressing mode; however, the MOVB, MOVE, and PIXT instructions each support several addressing modes. The following subsections describe the TMS34010 addressing modes.

12.2.1 Immediate Addressing

In this addressing mode, the source operand may be one of the following:

- A 16-bit immediate value (designated as IW)
- A 32-bit immediate value (designated as IL)
- A constant (designated as K)

Figure 12-1 shows an example of the `MOVI <IL>,<Rd>` instruction. A 32-bit immediate value, >FC00, is loaded into the destination register, A3.

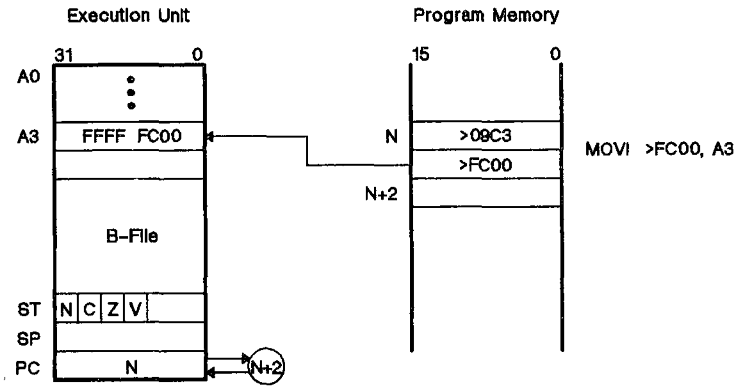


Figure 12-1. Immediate Addressing Mode

12.2.2 Indirect XY

A source operand or a destination operand can be specified using this addressing mode.

- **Rs.XY* – The register contains the XY address of the data.
- **Rd.XY* – The register contains the XY address where the data will be moved.

12.2.3 Absolute Addressing

A source operand or a destination operand can be specified as an absolute address.

- *@SAddress* – The specified address contains the data.
- *@DAddress* – The data will be moved into the specified address.

Figure 12-2 shows an example of the `MOVB @<SAddress>,<Rd>` instruction. In this example, the symbol *FADDR* represents a memory address; the data at this address are loaded into register A4.

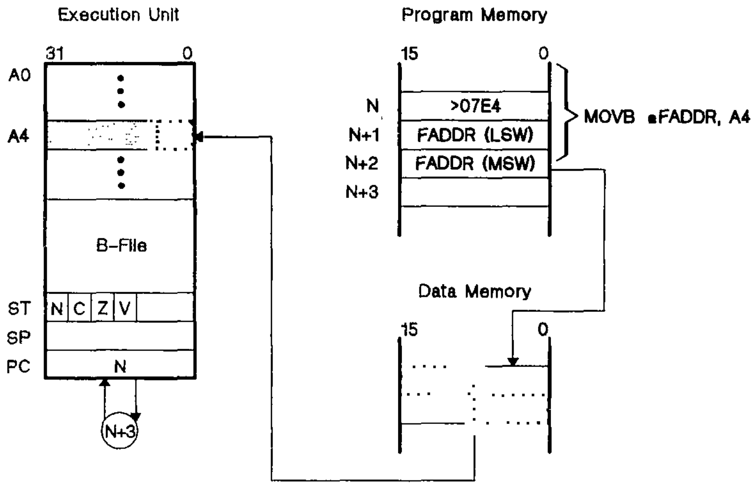


Figure 12-2. Absolute Addressing Mode

12.2.4 Register Direct

A source operand or a destination operand can be specified using register direct addressing mode.

- *Rs* – The source register contains the data.
- *Rd* – The data will be moved into the destination register.

Figure 12-3 shows an example of the `MOVE <Rs>,<Rd>` instruction. The contents of the source register, A3, are moved into the destination register, B2.

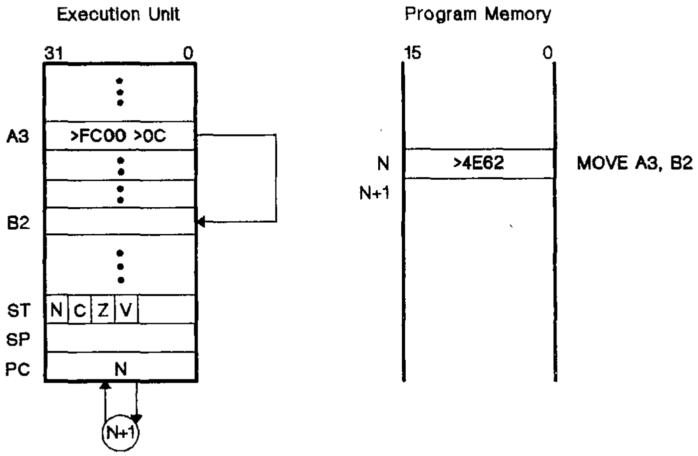


Figure 12-3. Register Direct Addressing Mode

12.2.5 Register Indirect

A source operand or a destination operand can be specified using register indirect addressing mode.

- **Rs* – The register contains the address of the data.
- **Rd* – The register contains the address where the data will be moved.

Figure 12-4 shows an example of the `MOVE <Rs>,*<Rd>,[<F>]` instruction. Register A4 contains the source operand. Register A3 contains an address (represented by the symbol *FADDR*) where the data in A4 will be moved.

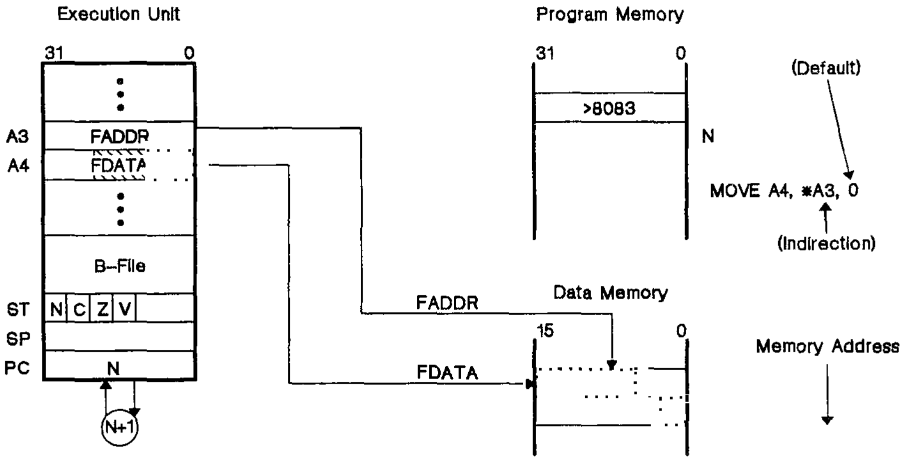


Figure 12-4. Register Indirect Addressing Mode

12.2.6 Register Indirect with Displacement

A source operand or a destination operand can be specified using this addressing mode.

- **Rs(Displacement)* – The address of the data is found by adding the register contents to the signed displacement.
- **Rd(Displacement)* – The data will be moved to the address specified by the sum register contents and the signed displacement.

Figure 12-5 shows an example of the `MOVE <Rs>,*<Rd>(<Displacement>)` instruction. Register A4 contains the source operand. Register A3 contains an address (represented by the symbol *FADDR*). The displacement, 16, is added to FADDR, to point to the location where the data in A4 will be moved. FS0 contains the field size.

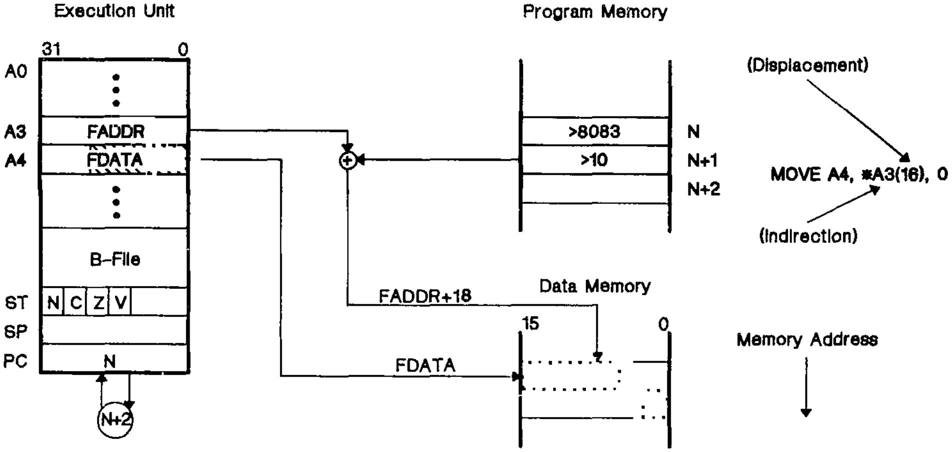


Figure 12-5. Register Indirect with Displacement Addressing Mode

12.2.7 Register Indirect with Predecrement

A source operand or a destination operand can be specified using this addressing mode.

- *-*Rs* – The address of the data is found by decrementing the register contents by the field size of the move.
- *-*Rd* – The data will stored at the address found by decrementing the register contents by the field size of the move.

Figure 12-6 shows an example of the `MOVE <Rs>,*-<Rd>` instruction. Register A4 contains the source operand. Register A3 contains an address (represented by the symbol *FADDR*). This address is decremented by the field size of the move, so that it points to the location where the data in A4 will be moved. FS1 contains the field size.

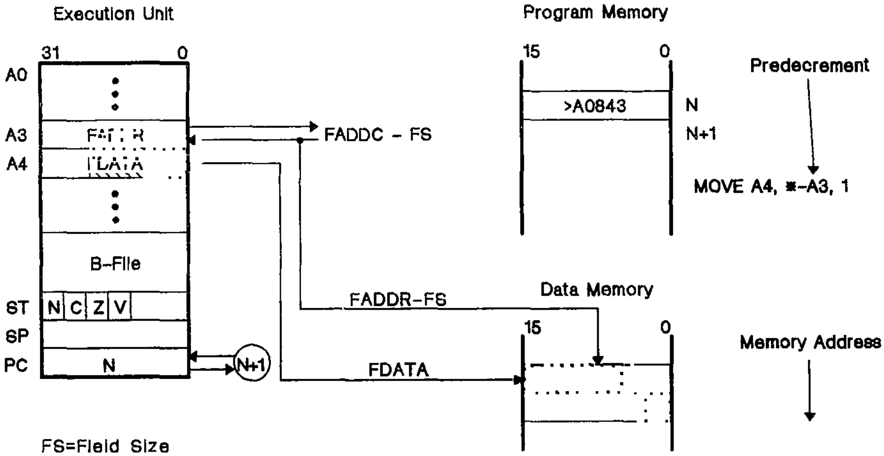


Figure 12-6. Register Indirect with Predecrement Addressing Mode

12.2.8 Register Indirect with Postincrement

A source operand or a destination operand can be specified using this addressing mode.

- **Rs+* – The register contains the address of the data. The register contents are incremented after the move.
- **Rd+* – The register contains the address where the data will be moved. The register contents are incremented after the move.

Figure 12-7 shows an example of the `MOVE <Rs>,*-<Rd>` instruction. Register A4 contains the source operand. Register A3 contains an address (represented by *FADDR*) where the data in A4 will be moved. The register contents are incremented after the move. FS0 contains the field size.

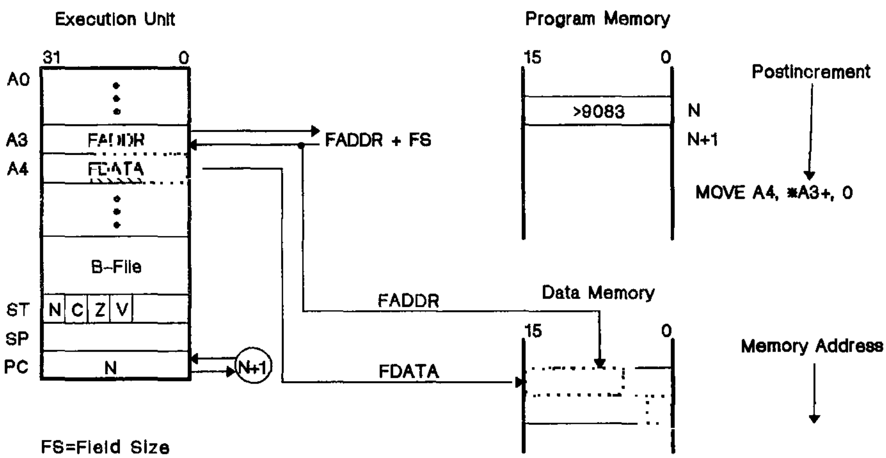


Figure 12-7. Register Indirect with Postincrement Addressing Mode

12.3 Move Instructions Summary

The move instructions use the GSP's bit-addressing and field operation capabilities to provide flexible memory management. All memory addresses for move operations are bit addresses. When a field is moved from memory to a register. Register bits to the left of the field are filled with either 0s or the sign bit, depending on the field extension mode. When a field is moved to memory from a register, the data for the field is assumed to be right justified within the register, and the bits to the left of the field are ignored. Table 12-2 summarizes the GSP move instructions.

Table 12-2. Summary of Move Instructions

Move Type	Mnemonic	Description
Register	MOVE	Move register to register
Constant	MOVK	Move constant (5 bits)
	MOVI	Move immediate (16 bits)
	MOVI	Move immediate (32 bits)
XY	MOVX	Move 16 LSBs of register (X half)
	MOVY	Move 16 MSBs of register (Y half)
Multiple Register	MMFM	Move multiple registers from memory
	MMTM	Move multiple registers to memory
Byte	MOVB	Move byte (8 bits, 9 addressing modes)
Field	MOVE	Move field to/from memory/register (15 addressing modes)

12.3.1 Register-to-Register Moves

The register-to-register MOVE instruction moves data directly between register files A and B. This is a 32-bit move; the entire contents of the destination register are replaced.

12.3.2 Constant-to-Register Moves

The MOVK and MOVI instructions load a register with a constant value. MOVK places a zero-extended value of 1 to 32 in the register. MOVI has two modes, 16-bit and 32-bit. The 32-bit MOVI uses two extension words which explicitly define the value to be stored in the register. The extension word for the 16-bit MOVI contains a value which is sign extended to 32 bits when moved into the register. Use the CLR instruction to store 0 in a register.

12.3.3 X and Y Register Moves

The MOVX and MOVY instructions move the X and Y halves, respectively; the other half ot the destination register is not affected. These are 16-bit moves within the register file. XY addressing is discussed in Section 4.

12.3.4 Multiple Register Moves

Multiple-register moves save and restore select members of up to an entire *file* of registers to memory. A 16-bit mask specifies which of the 16 registers in the designated file are to be moved to or from memory. One register from the selected file acts as a pointer register for the move. Any of the registers in the file, including the SP, may be used as the pointer register. The selected registers are input as a list; the assembler checks that they and the pointer register are all in the same file. The pointer register contains a bit address for the register "stack." The stacking operation follows the same conventions as the system stack, growing in the direction of lower memory. If the SP is used, both register files may be moved to the same stack area (since SP may be accessed from both files). MMTM moves multiple registers to the stack while MMFM moves them from memory back to the register file.

12.3.5 Byte Moves

Byte moves are special 8-bit cases of the field moves described in Section 12.3.6. Byte moves are implicitly 8-bit moves. They transfer data:

- From memory to a register (using field extraction),
- From a register to memory (using field insertion),
  or
- From memory to memory (using field extraction and field insertion).

A byte can begin on any bit boundary within a word. When a byte is moved from memory to a general-purpose register, it is right justified within the register so that the LSB of the byte coincides with the rightmost bit (bit 0) of the register. The byte is sign extended to fill the 24 MSBs of the register.

Table 12-3 lists the possible combinations of source and destination addressing modes for MOVBs.

Table 12-3. MOVB Addressing Modes

Source Addressing Mode	Destination Addressing Mode			
	Rd	*Rd	*Rd(disp)	@Address
Rs		●	●	●
*Rs	●	●		
*Rs(Disp)	●		●	
@Address	●			●

Note: The ● symbol indicates a valid operation; a blank box indicates an invalid operation.

Sequences of byte-move operations can be expected to execute more efficiently if the byte address points to an even 8-bit boundary within memory. This occurs when the three LSBs of the 32-bit starting address of the byte are 0. A byte that straddles a word boundary requires twice as many memory cycles to access.

12.3.6 Field Moves

A field is a configurable data structure in memory. It is identified by two parameters - size and data address. A field's length can be defined to be any value from 1 to 32 bits. Field moves manipulate arbitrarily-sized data fields in memory and the register file.

- Field data in *memory* is addressed by its bit address and is treated as a string of contiguous bits; it may start at any bit address in memory.
- Field data in *the register file* is right justified in the register; the LSB of the field is stored in the LSB of the register.

When field data is moved into a register, it is right justified within the register. The register bits to the left of the field are all 1s or all 0s, depending on the values of both the appropriate FE (field extension) bit in the status register, and sign bit (MSB) of the field. If FE=1, the field is sign extended; if FE=0, the field is zero extended. When data is moved from a register, these non-field bits of the register are ignored.

Fields are transferred between the general-purpose registers and memory by means of the memory-to-register and register-to-memory move instructions. Fields are transferred from one memory location to another via the memory-to-memory move instructions.

Table 12-4 lists the possible combinations of source and destination addressing modes for MOVEs.

Table 12-4. Field Move Addressing Modes

Source Addressing Mode	Destination Addressing Mode					
	Rd	*Rd	*Rd+	-*Rd	*Rd(disp)	@Address
Rs		●	●	●	●	●
*Rs	●	●				
*Rs+	●		●			
-*Rs	●			●		
*Rs(Disp)	●		●		●	
@Addr	●		●			●

Note: The ● symbol indicates a valid operation; a blank box indicates an invalid operation.

Two field sizes are simultaneously available for field moves. The lengths of fields 0 and 1 are defined by two 5-bit fields in the status register, FS0 and FS1. The status register also contains the FE0 and FE1 parameters, which define the field extension properties of the data when it is moved into a register.

The SETF instruction specifies the size and signed/unsigned condition of either field 0 or 1 by placing this data in one of two 6-bit fields located in the

status register. One bit specifies sign/zero extension, and five bits store the field size (in bits).

The EXGF instruction may also set either of the two field types, while preserving a copy of the previous definition.

The address of a field points to its least significant bit. A field can begin at an arbitrary bit address in memory. Field data addresses for particular moves are derived from values in registers and extension words following the instruction. Field moves transfer data:

- From memory to a register (using field extraction),
- From a register to memory (using field insertion),
  or
- From memory to memory (using field extraction and field insertion).

12.3.6.1 Register-to-Memory Field Moves

Figure 12-8 illustrates the register-to-memory move operation. In this type of move, the source register contains the right-justified field data (width is specified by the field size). The destination memory location is the bit position pointed to by the destination memory address. The address consists of a portion defining the starting word in which the field is to be written and an offset into that word, the bit address. Depending on the bit address within this word and the field size, the destination location may extend into two or more words. The field size for the move is one of two indirect values stored in ST, as selected by the programmer. The field extension bit is not used.

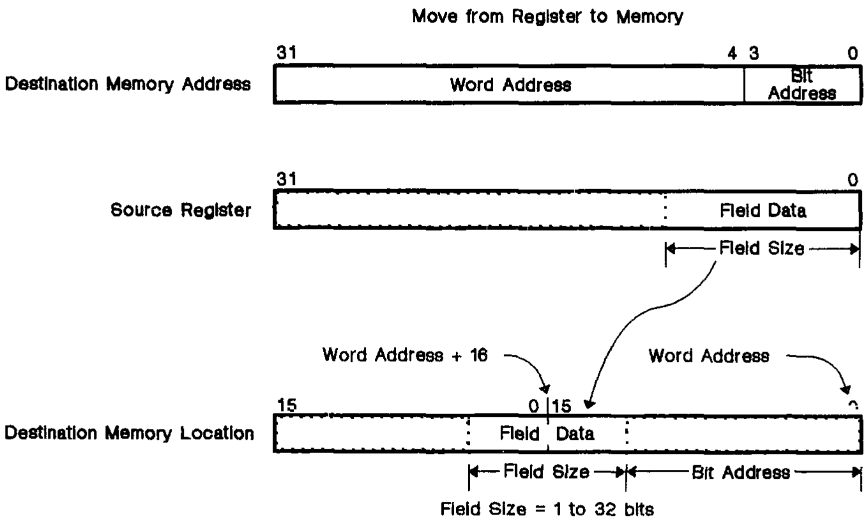


Figure 12-8. Register-to-Memory Moves

12.3.6.2 Memory-to-Register Field Moves

Figure 12-9 shows the memory-to-register move operation. The source memory location is the bit position pointed to by the source memory address. The address consists of a portion defining the starting word in which the field is to be written and an offset into that word, the bit address. Depending on the bit address within this word and the field size, the source location may extend into two or more words. After the move, the destination register LSBs contain the right-justified field data (width is specified by the field size). The MSBs of the register contain either all 1s or all 0s. If the sign extension bit FE0 or FE1 associated with the field size selected is 0, the MSBs are 0s. If the sign extension bit selected is 1, the MSBs contain the value of the sign bit of the field data (its MSB). The field size for the move is one of two indirect values stored in ST, as selected by the programmer.

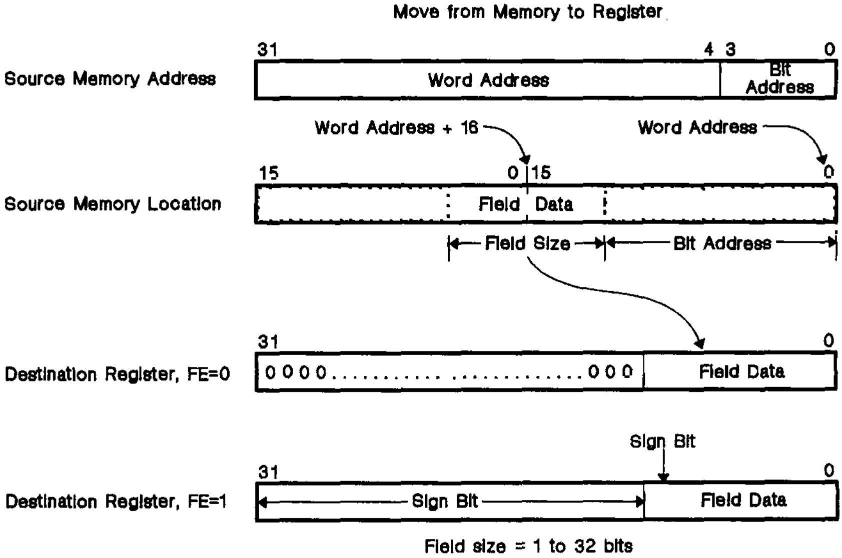


Figure 12-9. Memory-to-Register Moves

12.3.6.3 Memory-to-Memory Field Moves

Figure 12-10 shows a memory-to-memory field move operation. The source memory location is the bit position pointed to by the source memory address. The destination memory location is the bit position pointed to by the destination memory address. Depending on the bit addresses within the respective words and the field size, either the source location or destination locations may extend into two or more words. After the move, the destination location contains the field data from the source memory location. The field size for the move is one of two indirect values stored in ST, as selected by the programmer. The field extension bit is not used.

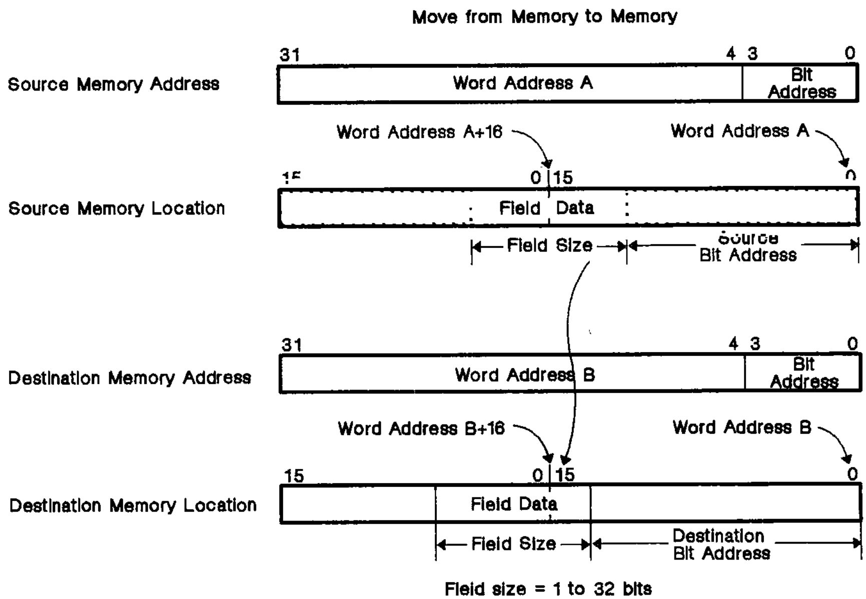


Figure 12-10. Memory-to-Memory Moves

12.4 PIXBLT Instructions Summary

The TMS34010 supports 6 different PIXBLT instructions. PIXBLTs vary according to the format of the source and destination pixel blocks. Table 12-5 summarizes the PIXBLT instructions.

Table 12-5. PIXBLT Instruction Summary

Syntax	Formats	Page
PIXBLT B,L	Binary to linear	12-157
PIXBLT B,XY	Binary to XY	12-162
PIXBLT L,L	Linear to linear	12-169
PIXBLT L,XY	Linear to XY	12-175
PIXBLT XY,L	XY to linear	12-181
PIXBLT XY,XY	XY to XY	12-186

12.5 PIXT Instructions Summary

The PIXT instructions move single pixels. The pixel may originate from a register or a memory location, and may be moved to a register or a memory location. There are 6 variations of the PIXT instruction; each uses a different combination of the addressing modes described in Section 12.2.

Table 12-6 lists the possible combinations of source and destination addressing modes for PIXTs.

Table 12-6. PIXT Addressing Modes

Source Addressing Mode	Destination Addressing Mode		
	Rd	*Rd	*Rd.XY
Rs		●	●
*Rs	●	●	
*Rs.XY		●	●

Note: The ● symbol indicates a valid operation; a blank box indicates an invalid operation.

Table 12-7. TMS34010 Instruction Set Summary

Graphics Instructions			
Syntax and Description	**Words**	**Machine States**	**16-Bit Opcode MSB LSB**
ADDXY Rs,Rd Add registers in XY mode	1	1,4	1110 000S SSSR DDDD
CMPXY Rd,Rd Compare X and Y halves of registers	1	3,6	1110 010S SSSR DDDD
CPW Rs,Rd Compare point to window	1	1,4	1110 011S SSSR DDDD
CVXYL Rs,Rd Convert XY address to linear address	1	3,6	1110 100S SSSR DDDD
DRAV Rs,Rd Draw and advance	1	†	1111 011S SSSR DDDD
FILL L Fill array with processed pixels, linear	1	‡	0000 1111 1100 0000
FILL XY Fill array with processed pixels, XY	1	‡	0000 1111 1110 0000
MOVX Rs,Rd Move X half of register	1	1,4	1110 110S SSSR DDDD
MOVY Rs,Rd Move Y half of register	1	1,4	1110 111S SSSR DDDD
PIXBLT B,L Pixel block transfer, binary to linear	1	‡‡	0000 1111 1000 0000
PIXBLT B,XY Pixel block transfer and expand, binary to XY	1	‡‡	0000 1111 1010 0000
PIXBLT L,L Pixel block transfer, linear to linear	1	§	0000 1111 0000 0000
PIXBLT L,XY Pixel block transfer, linear to XY	1	§	0000 1111 0010 0000
PIXBLT XY,L Pixel block transfer, XY to linear	1	§	0000 1111 0100 0000
PIXBLT XY,XY Pixel block transfer, XY to XY	1	§	0000 1111 0110 0000
PIXT Rs,*Rd Pixel transfer, register to indirect	1	†	1111 100S SSSR DDDD
PIXT Rs,*Rd.XY Pixel transfer, register to indirect XY	1	†	1111 000S SSSR DDDD
PIXT *Rs,Rd Pixel transfer, indirect to register	1	†	1111 101S SSSR DDDD
PIXT *Rs,*Rd Pixel transfer, indirect to indirect	1	†	1111 110S SSSR DDDD
PIXT *Rs.XY,Rd Pixel transfer, indirect XY to register	1	†	1111 001S SSSR DDDD
PIXT *Rs.XY,*Rd.XY Pixel transfer, indirect XY to indirect XY	1	†	1111 010S SSSR DDDD
SUBXY Rs,Rd Subtract registers in XY mode	1	1,4	1110 001S SSSR DDDD
LINE Z Line draw	1	Δ	1101 1111 Z001 1010

† See instruction
‡ See Section 13.3, FILL Instructions Timing
‡‡ See Section 13.5, PIXBLT Expand Instructions Timing
§ See Section 13.4, PIXBLT Instructions Timing
Δ See Section 13.6, The LINE Instruction Timing

Table 12-7. TMS34010 Instruction Set Summary (Continued)

Move Instructions			
Syntax and Description	**Words**	**Machine States**	**16-Bit Opcode MSB LSB**
MOVB Rs,*Rd Move byte, register to indirect	1	¶	1000 110S SSSR DDDD
MOVB *Rs,Rd Move byte, indirect to register	1	¶	1000 111S SSSR DDDD
MOVB *Rs,*Rd Move byte, indirect to indirect	1	¶	1001 110S SSSR DDDD
MOVB *Rs,*Rd(Disp) Move byte, register to indirect with displacement	2	¶	1010 110S SSSR DDDD
MOVB *Rs(Disp),Rd Move byte, indirect with displacement to register	2	¶	1010 111S SSSR DDDD
MOVB *Rs(Disp),*Rd(Disp) Move byte, indirect with displacement to indirect with displacement	3	¶	1011 110S SSSR DDDD
MOVB Rs,@DAddress Move byte, register to absolute	3	¶	0000 0101 111R SSSS
MOVB @SAddress,Rd Move byte, absolute to register	3	¶	0000 0111 111R DDDD
MOVB @SAddress,@DAddress Move byte, absolute to absolute	5	¶	0000 0011 0100 0000
MOVE Rs,Rd Move register to register	1	1,4	0100 11MS SSSR DDDD
MOVE Rs,*Rd,F Move field, register to indirect	1	¶	1000 00FS SSSR DDDD
MOVE Rs,-*Rd,F Move field, register to indirect (predecrement)	1	¶	1010 00FS SSSR DDDD
MOVE Rs,*Rd+,F Move field, register to indirect (postincrement)	1	¶	1001 00FS SSSR DDDD
MOVE *Rs,Rd,F Move field, indirect to register	1	¶	1000 01FS SSSR DDDD
MOVE -*Rs,Rd,F Move field, indirect (predecrement) to register	1	¶	1010 01FS SSSR DDDD
MOVE *Rs+,Rd,F Move field, indirect (postincrement) to register	1	¶	1001 01FS SSSR DDDD
MOVE *Rs,*Rd,F Move field, indirect to indirect	1	¶	1000 10FS SSSR DDDD
MOVE -*Rs,-*Rd,F Move field, indirect (predecrement) to indirect (predecrement)	1	¶	1010 10FS SSSR DDDD
MOVE *Rs+,*Rd+,F Move field, indirect (postincrement) to indirect (postincrement)	1	¶	1001 10FS SSSR DDDD
MOVE Rs,*Rd(Disp),F Move field, register to indirect with displacement	2	¶	1011 00FS SSSR DDDD
MOVE *Rs(Disp),Rd,F Move field, indirect with displacement to register	2	¶	1011 01FS SSSR DDDD

¶ See Section 13.2, MOVE and MOVB Instructions Timing

Table 12-7. TMS34010 Instruction Set Summary (Continued)

Move Instructions (Continued)			
Syntax and Description	**Words**	**Machine States**	**16-Bit Opcode MSB LSB**
MOVE *Rs(Disp),*Rd+,F Move field, indirect with displacement to indirect (postincrement)	2	¶	1101 00FS SSSR DDDD
MOVE *Rs(Disp),*Rd(Disp),F Move field, indirect with displacement to indirect with displacement	3	¶	1011 10FS SSSR DDDD
MOVE Rs,@DAddress,F Move field, register to absolute	3	¶	0000 01F1 100R DDDD
MOVE @SAddress,Rd,F Move field, absolute to register	3	¶	0000 01F1 101R DDDD
MOVE @SAddress,*Rd+,F Move field, absolute to indirect (postincrement)	3	¶	1101 01F0 000R DDDD
MOVE @SAddress,@DAddress,F Move field, absolute to absolute	5	¶	0000 01F1 1100 DDDD

General Instructions			
Syntax and Description	**Words**	**Machine States**	**16-Bit Opcode MSB LSB**
ABS Rd Store absolute value	1	1,4	0000 0011 100R DDDD
ADD Rs,Rd Add registers	1	1,4	0100 000S SSSR DDDD
ADDC Rs,Rd Add registers with carry	1	1,4	0100 001S SSSR DDDD
ADDI IW,Rd Add immediate (16 bits)	2	2,8	0000 1011 000R DDDD
ADDI IL,Rd Add immediate (32 bits)	3	3,12	0000 1011 001R DDDD
ADDK K,Rd Add constant (5 bits)	1	1,4	0001 00KK KKKR DDDD
AND Rs,Rd AND registers	1	1,4	0101 000S SSSR DDDD
ANDI IL,Rd AND immediate (32 bits)	3	3,12	0000 1011 100R DDDD
ANDN Rs,Rd AND register with complement	1	1,4	0101 001S SSSR DDDD
ANDNI IL,Rd AND not immediate (32 bits)	3	3,12	0000 1011 100R DDDD
BTST K,Rd Test register bit, constant	1	1,4	0001 11KK KKKR DDDD
BTST Rs,Rd Test register bit, register	1	2,5	0100 101S SSSR DDDD
CLR Rd Clear register	1	1,4	0101 011D DDDR DDDD
CLRC Clear carry	1	1,4	0000 0011 0010 0000
CMP Rs,Rd Compare registers	1	1,4	0000 1011 010R DDDD

¶ See Section 13.2, MOVE and MOVB Instructions Timing

Table 12-7. TMS34010 Instruction Set Summary (Continued)

General Instructions (Continued)			
Syntax and Description	Words	Machine States	16-Bit Opcode MSB LSB
CMPI IW,Rd Compare immediate (16 bits)	2	2,8	0000 1011 010R DDDD
CMPI IL,Rd Compare immediate (32 bits)	3	3,12	0000 1011 011R DDDD
DEC Rd Decrement register	1	1,4	0001 0100 001R DDDD
DINT Disable interrupts	1	3,6	0000 0011 0110 0000
DIVS Rs,Rd Divide registers signed	1	40,43 39,42 Δ	0101 100S SSSR DDDD
DIVU Rs,Rd Divide registers unsigned	1	37,40	0101 101S SSSR DDDD
EINT Enable interrupts	1	3,6	0000 1101 0110 0000
EXGF Rd,F Exchange field size	1	1,4	1101 01F1 000R DDDD
LMO Rs,Rd Leftmost one	1	1,4	0110 101S SSSR DDDD
MMFM Rs,List Move multiple registers from memory	2	†	0000 1001 101R DDDD
MMTM Rs,List Move multiple registers to memory	2	†	0000 1001 100R DDDD
MODS Rs,Rd Modulus signed	1	40,43	0110 110S SSSR DDDD
MODU Rs,Rd Modulus unsigned	1	35,38	0110 111S SSSR DDDD
MOVI IW,Rd Move immediate (16 bits)	2	2,8	0000 1001 110R DDDD
MOVI IL,Rd Move immediate (32 bits)	3	3,12	0000 1001 111R DDDD
MOVK K,Rd Move constant (5 bits)	1	1,4	0001 10KK KKKR DDDD
MPYS Rs,Rd Multiply registers (signed)	1	20,23	0101 110S SSSR DDDD
MPYU Rs,Rd Multiply registers (unsigned)	1	21,24	0101 111S SSSR DDDD
NEG Rd Negate register	1	1,4	0000 0011 101R DDDD
NEGB Rd Negate register with borrow	1	1,4	0000 0011 110R DDDD
NOP No operation	1	1,4	0000 0011 0000 0000
NOT Rd Complement register	1	1,4	0000 0011 111R DDDD

† See instruction
‡ If F=1, add 1 to cycle time
Δ Rd even/Rd odd

Table 12-7. TMS34010 Instruction Set Summary (Continued)

General Instructions (Continued)			
Syntax and Description	**Words**	**Machine States**	**16-Bit Opcode** MSB LSB
OR Rs,Rd OR registers	1	1,4	0101 010S SSSR DDDD
ORI L,Rd OR immediate (32 bits)	3	3,12	0000 1011 101R DDDD
RL K,Rd Rotate left, constant	1	1,4	0011 00KK KKKR DDDD
RL Rs,Rd Rotate left, register	1	1,4	0110 10SS SSSR DDDD
SETC Set carry	1	1,4	0000 1101 1110 0000
SETF FS,FE,F Set field parameters	1	1,4 2,5 ‡	0000 01F1 01FS SSSS
SEXT Rd,F Sign extend to long	1	3,6	0000 01F1 000R DDDD
SLA K,Rd Shift left arithmetic, constant	1	3,6	0010 00KK KKKR DDDD
SLA Rs,Rd Shift left arithmetic, register	1	3,6	0110 000S SSSR DDDD
SLL K,Rd Shift left logical, constant	1	1,4	0010 01KK KKKR DDDD
SLL Rs,Rd Shift left logical, register	1	1,4	0110 001S SSSR DDDD
SRA K,Rd Shift right arithmetic, constant	1	1,4	0010 10KK KKKR DDDD
SRA Rs,Rd Shift right arithmetic, register	1	1,4	0110 010S SSSR DDDD
SRL K,Rd Shift right logical, constant	1	1,4	0010 11KK KKKR DDDD
SRL Rs,Rd Shift right logical, register	1	1,4	0110 011S SSSR DDDD
SUB Rs,Rd Subtract registers	1	1,4	0100 010S SSSR DDDD
SUBB Rs,Rd Subtract registers with borrow	1	1,4	0100 011S SSSR DDDD
SUBI IW,Rd Subtract immediate (16 bits)	2	2,8	0000 1011 111R DDDD
SUBI IL,Rd Subtract immediate (32 bits)	3	3,12	0000 1101 111R DDDD
SUBK K,Rd Subtract constant (5 bits)	1	1,4	0001 01KK KKKR DDDD
XOR Rs,Rd Exclusive OR registers	1	1,4	0101 011S SSSR DDDD
XORI IL,Rd E[illegible]ive OR immediate value (32 bits)	3	3,12	0000 1011 110D DDDD
Z[illegible] Rd,F Zero extend to long	1	1,4	0000 01F1 001R DDDD

† See instruction
‡ If F=1, add 1 to cycle time
Δ Rd even/Rd odd

Table 12-7. TMS34010 Instruction Set Summary (Concluded)

Program Control and Context Switching Instructions			
Syntax and Description	**Words**	**Machine States**	**16-Bit Opcode** **MSB LSB**
CALL Rs Call subroutine indirect	1	3+(3),9 3+(9),15 Θ	0000 1001 001R DDDD
CALLA Address Call subroutine address	3	4+(2),15 4+(8),21 Θ	0000 1101 0101 1111
CALLR Address Call subroutine relative	2	3+(2),11 3+(8),17 Θ	0000 1101 0011 1111
DSJ Rd,Address Decrement register and skip jump	2	3,9 2,8 Π	0000 1101 100R DDDD
DSJEQ Rd,Address Conditionally decrement register and skip jump	2	3,9 2,8 Π	0000 1101 101R DDDD
DSJNE Rd,Address Conditionally decrement register and skip jump	2	3,9 2,8 Π	0000 1101 110R DDDD
DSJS Rd,Address Decrement register and skip jump short	1	2,5 3,6 Π	0011 1DKK KKKR DDDD
EMU Initiate emulation	1	6,9	0000 0001 0000 0000
EXGPC Rd Exchange program counter with register	1	2,5	0000 0001 001R DDDD
GETPC Rd Get program counter into register	1	1,4	0000 0001 010R DDDD
GETST Rd Get status register into register	1	1,4	0000 0001 100R DDDD
JAcc Address Jump absolute conditional	3	3,6 4,7 Π	1100 code 1000 0000
JRcc Address Jump relative conditional	2	3,6 1,4 Π	1100 code 0000 0000
JRcc Address Jump relative conditional short	1	2,5 2,5 Π	1100 code xxxx xxxx
JUMP Rs Jump indirect	1	2,5	0000 0001 011R DDDD
POPST Pop status register from stack	1	8,11 10,13 Θ	0000 0001 1100 0000
PUSHST Push status register onto stack	1	2+(3),8 2+(8),13 Θ	0000 0001 1110 0000
PUTST Rs Copy register into status	1	3,6	0000 0001 101R DDDD
RETI Return from interrupt	1	11,14 15,18 Φ	0000 1001 0100 0000
RETS [N] Return from subroutine	1	7,10 9,12 Φ	0000 1001 011N NNNN
TRAP N Software interrupt	1	16,19 30,33 Θ	0000 1001 000N NNNN

Θ SP aligned/SP nonaligned
Π Jump/no jump
Φ Stack aligned/stack nonaligned

Syntax This line shows you how to enter an instruction. Here are some sample syntaxes:

- **EXAMPLE** *<source operand>,<destination operand>*

  If an operand is enclosed in angle brackets (< and >), substitute actual source and destination operands (such as a register or constant) for the text that is shown.

- **EXAMPLE** B,XY

  If an operand is **not** enclosed in angle brackets, then enter it as shown. In this example, you would actually enter `EXAMPLE B,XY`.

- **EXAMPLE** *<source operand>*[,*< destination operand>*]

  If an operand is enclosed in square brackets ([ ]), then the operand is optional. (Do not enter the brackets.) This example could be entered as `EXAMPLE source operand, destination operand` **or** as `EXAMPLE source operand`.

Execution This section describes instruction execution. The general form is:

<operand> operator <operand> → <operand>

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	<source opd>				R	<destination opd>			

This section displays the contents of the instruction word.

Operands This section describes any instruction operands and elements of the preceding opcode format. Any assembler exception handling for operands may be described here.

Fields This line discusses any fields in the opcode that are not explicit operands.

Description This section describes the instruction execution and its effect on the rest of the processor or memory contents. Any constraints on the operands imposed by the GSP or the assembler are also described here. Special instruction applications may follow the description.

Implied Operands

This section describes any operands which are implicit inputs to the instruction. These operands are usually B file registers and I/O registers and are described in detail in Sections 5 and 6. You must load these registers with appropriate values before instruction execution.

B File Registers			
Register	**Name**	**Format**	**Description**
. .	. .	. .	. .

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
. .	. .	. .

Special Graphics Topics

Graphics instructions (DRAV, PIXBLTs, etc.) may present special topics of discussion under the following headings:

- Source Array
- Source Expansion
- Destination Array
- Pixel Processing
- Window Checking
- Transparency
- Corner Adjust
- Plane Mask
- Shift Register Transfers

Interrupts Discusses the effects of possible interrupts.

Words Specifies the number of memory words required to store the instruction and its extension words.

Machine States Cache resident + (Hidden cycles), Cache disabled

Specifies instruction cycle timing for the instruction. Not all instructions have hidden cycles. Section 13, Instruction Timings, provides a complete explanation of instruction timing.

Status Bits

N Describes the instruction's effects on the sign bit.
C Describes the instruction's effects on the carry bit.
Z Describes the instruction's effects on the zero bit.
V Describes the instruction's effects on the overflow bit.

Examples Each instruction description contains sample code, and shows the effects of the code on memory and/or registers.

Syntax ABS <*Rd*>

Execution |(Rd)| → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	0	0	R	Rd			

Description ABS stores the absolute value of the contents of the destination register back into the destination register. This is accomplished by subtracting the destination register data from 0 and storing it if status bit N indicates that the result is positive. If the result of the subtraction is negative, then the original contents of the destination register are retained.

Words 1

Machine States 1,4

Status Bits

- **N** *1* if the original data is positive, *0* otherwise. This status bit is the inverse of its normal function; it is the output of the subtract-from-0 operation.
- **C** Unaffected
- **Z** *1* if the original data is 0, *0* otherwise.
- **V** *1* if there is an overflow, *0* otherwise. An overflow occurs if Rd contains >8000 0000 (>8000 0000 is returned).

Examples

Code	Before	After	
	A1	NCZV	A1
ABS A1	>7FFF FFFF	1x00	>7FFF FFFF
ABS A1	>FFFF FFFF	0x00	>0000 0001
ABS A1	>8000 0000	1x01	>8000 0000
ABS A1	>8000 0001	0x00	>7FFF FFFF
ABS A1	>0000 0001	1x00	>0000 0001
ABS A1	>0000 0000	0x10	>0000 0000
ABS A1	>FFFA 0011	0x00	>0005 FFEF

Syntax ADD <*Rs*>,<*Rd*>

Execution (Rs) + (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	Rs				R	Rd			

Description ADD adds the contents of the source register to the contents of the destination register; the result is stored in the destination register.

Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the ADDC instruction.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a carry, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A1	Before A0	After NCZV	After A0
ADD A1,A0	>FFFF FFFF	>FFFF FFFF	1100	>FFFF FFFE
ADD A1,A0	>FFFF FFFF	>0000 0001	0110	>0000 0000
ADD A1,A0	>FFFF FFFF	>0000 0002	0100	>0000 0001
ADD A1,A0	>FFFF FFFF	>8000 0000	0101	>7FFF FFFF
ADD A1,A0	>FFFF FFFF	>8000 0001	1100	>8000 0000
ADD A1,A0	>7FFF FFFF	>8000 0001	0110	>0000 0000
ADD A1,A0	>7FFF FFFF	>8000 0000	1000	>FFFF FFFF
ADD A1,A0	>7FFF FFFF	>0000 0001	1001	>8000 0000
ADD A1,A0	>0000 0002	>0000 0002	0000	>0000 0004

Add Register with Carry

Syntax **ADDC** *<Rs>,<Rd>*

Execution (Rs) + (Rd) + (C) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	Rs				R	Rd			

Description ADDC adds the contents of the source register and the status carry bit to the contents of the destination register; the result is stored in the destination register. Note that the status bits are set on the collective add.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
N *1* if the result is negative, *0* otherwise.
C *1* if there is a carry, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before C	Before A1	Before A0	After NCZV	After A0
`ADDC A1,A0`	1	>FFFF FFFF	>FFFF FFFF	1100	>FFFF FFFF
`ADDC A1,A0`	1	>FFFF FFFF	>0000 0001	0100	>0000 0001
`ADDC A1,A0`	1	>FFFF FFFF	>0000 0002	0100	>0000 0002
`ADDC A1,A0`	1	>FFFF FFFF	>8000 0000	1100	>8000 0000
`ADDC A1,A0`	1	>FFFF FFFF	>8000 0001	1100	>8000 0001
`ADDC A1,A0`	1	>FFFF FFFF	>8000 0001	0100	>8000 0001
`ADDC A1,A0`	1	>FFFF FFFF	>8000 0000	0110	>0000 0000
`ADDC A1,A0`	1	>7FFF FFFF	>0000 0001	1001	>8000 0001
`ADDC A1,A0`	1	>0000 0002	>0000 0002	0000	>0000 0005
`ADDC A1,A0`	0	>FFFF FFFF	>FFFF FFFF	1100	>FFFF FFFE
`ADDC A1,A0`	0	>FFFF FFFF	>0000 0001	0110	>0000 0000
`ADDC A1,A0`	0	>FFFF FFFF	>0000 0002	0100	>0000 0001
`ADDC A1,A0`	0	>FFFF FFFF	>8000 0000	0101	>7FFF FFFF
`ADDC A1,A0`	0	>FFFF FFFF	>8000 0001	1100	>8000 0000
`ADDC A1,A0`	0	>7FFF FFFF	>8000 0001	0110	>0000 0000
`ADDC A1,A0`	0	>7FFF FFFF	>8000 0000	1000	>FFFF FFFF
`ADDC A1,A0`	0	>7FFF FFFF	>0000 0001	1001	>8000 0000
`ADDC A1,A0`	0	>0000 0002	>0000 0002	0000	>0000 0004

Syntax **ADDI** <*IW*>,<*Rd*>[,W]

Execution IW + (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	0	0	R	Rd			
IW															

Operands **IW** is a 16-bit, sign-extended immediate value.

Description ADDI adds the sign-extended, 16-bit immediate value to the contents of the destination register; the result is stored in the destination register.

The assembler will use the short (16-bit) add if the immediate value has been previously defined and is in the range -32,768 ≤ IW ≤ 32,767. You can force the assembler to use the short form by following the instruction with **W**:

```
ADDI  <IW>,<Rd>,W
```

If the IW value is outside the legal range, the assembler will discard all but the 16 LSBs and issue an appropriate warning message.

Multiple-precision arithmetic can be accomplished by using ADDI in conjunction with the ADDC instruction.

Words 2

Machine States 2,8

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a carry, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A0	After NCZV	After A0
`ADDI 1,A0`	>FFFF FFFF	0110	>0000 0000
`ADDI 2,A0`	>FFFF FFFF	0100	>0000 0001
`ADDI 1,A0`	>7FFF FFFF	1001	>8000 0000
`ADDI 2,A0`	>0000 0002	0000	>0000 0004
`ADDI 32767,A0`	>0000 0002	0000	>0000 8001
`ADDI >FFFF0010,A0,W`	>FFFF FFF0	0110	>0000 0000

Syntax **ADDI** *<IL>*,*<Rd>*[,**L**]

Execution IL + (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	0	1	R	Rd			
IL (LSW)															
IL (MSW)															

Operands **IL** is a 32-bit immediate value.

Description ADDI adds the 32-bit, signed immediate data to the contents of the destination register; the result is stored in the destination register.

The assembler will use the long (32-bit) ADDI if it cannot use the short form. You can force the assembler to use the long form by following the instruction with **L**:

```
ADDI <IL>,<Rd>,L
```

Words 3

Machine States 3,12

Status Bits
- **N** *1* if the result is negative, *0* otherwise.
- **C** *1* if there is a carry, *0* otherwise.
- **Z** *1* if the result is 0, *0* otherwise.
- **V** *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A0	After NCZV	After A0
`ADDI >FFFFFFFF,A0`	>FFFF FFFF	1100	>FFFF FFFE
`ADDI >80000000,A0`	>FFFF FFFF	0101	>7FFF FFFF
`ADDI >80000000,A0`	>7FFF FFFF	1000	>FFFF FFFF
`ADDI 32768,A0`	>7FFF FFFF	1001	>8000 7FFF
`ADDI 2,A0,L`	>FFFF FFFF	0100	>0000 0001

Syntax **ADDK** *<K>,<Rd>*

Execution K + (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	K					R	Rd			

Operands **K** is a constant from 1 to 32.

Description ADDK adds a 5-bit constant to the contents of the destination register; the result is stored in the destination register. The constant is treated as an unsigned number in the range 1–32, where K = 32 is converted to 0 in the opcode. The assembler will issue an error if you try to add 0 to a register.

Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the ADDC instruction.

Words 1

Machine States 1,4

Status Bits
N *1* if the result is negative, *0* otherwise.
C *1* if there is a carry, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A0	After NCZV	After A0
`ADDK 1,A0`	>FFFF FFFF	0110	>0000 0000
`ADDK 2,A0`	>FFFF FFFF	0100	>0000 0001
`ADDK 1,A0`	>7FFF FFFF	1001	>8000 0000
`ADDK 1,A0`	>8000 0000	1000	>8000 0001
`ADDK 32,A0`	>8000 0000	1000	>8000 0020
`ADDK 32,A0`	>0000 0002	0000	>0000 0022

Syntax **ADDXY** *<Rs>,<Rd>*

Execution (RsX) + (RdX) → RdX
(RsY) + (RdY) → RdY

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	Rs				R	Rd			

Description ADDXY adds the signed source X value to the signed destination X value, and adds the signed source Y value to the signed destination Y value. The result is stored in the destination register. The source and destination registers are treated as if they contained separate X and Y values. When they are added, the carry out from the lower (X) half of the register does not propagate into the upper (Y) half.

If you only want to add the X halves together, then the Y value of one of the operands must be 0 (the method for adding the Y halves is similar).

This instruction can be used for manipulating XY addresses in the register file and is particularly useful for incremental figure drawing.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
- **N** *1* if resulting X field is all 0s, *0* otherwise.
- **C** The sign bit of the Y half of the result.
- **Z** *1* if Y field is all 0s, *0* otherwise.
- **V** The sign bit of the X half of the result.

Examples

Code	Before A1	Before A0	After A0	NCZV
`ADDXY A1,A0`	>0000 0000	>0000 0000	>0000 0000	1010
`ADDXY A1,A0`	>0000 0000	>0000 0001	>0000 0001	0010
`ADDXY A1,A0`	>0000 0000	>0001 0000	>0001 0000	1000
`ADDXY A1,A0`	>0000 0000	>0001 0001	>0001 0001	0000
`ADDXY A1,A0`	>0000 FFFF	>0000 0001	>0000 0000	1010
`ADDXY A1,A0`	>0000 FFFF	>0001 0001	>0001 0000	1000
`ADDXY A1,A0`	>0000 FFFF	>0000 0002	>0000 0001	0010
`ADDXY A1,A0`	>0000 FFFF	>0001 0002	>0001 0001	0000
`ADDXY A1,A0`	>FFFF 0000	>0001 0000	>0000 0000	1010
`ADDXY A1,A0`	>FFFF 0000	>0001 0001	>0000 0001	0010
`ADDXY A1,A0`	>FFFF 0000	>0002 0000	>0001 0000	1000
`ADDXY A1,A0`	>FFFF 0000	>0002 0001	>0001 0001	0000
`ADDXY A1,A0`	>FFFF FFFF	>0001 0001	>0000 0000	1010
`ADDXY A1,A0`	>FFFF FFFF	>0001 0002	>0000 0001	0010
`ADDXY A1,A0`	>FFFF FFFF	>0002 0001	>0001 0000	1000
`ADDXY A1,A0`	>FFFF FFFF	>0002 0002	>0001 0001	0000

Syntax **AND** *<Rs>,<Rd>*

Execution (Rs) AND (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	Rs				R	Rd			

Description AND bitwise-ANDs the contents of the source register with the contents of the destination register; the result is stored in the destination register. The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before A1	Before A0	After NCZV	After A0
`AND A1,A0`	>FFFF FFFF	>FFFF FFFF	xx0x	>FFFF FFFF
`AND A1,A0`	>FFFF FFFF	>0000 0000	xx1x	>0000 0000
`AND A1,A0`	>0000 0000	>0000 0000	xx1x	>0000 0000
`AND A1,A0`	>AAAA AAAA	>5555 5555	xx1x	>0000 0000
`AND A1,A0`	>AAAA AAAA	>AAAA AAAA	xx0x	> AAAA AAAA
`AND A1,A0`	>5555 5555	>5555 5555	xx0x	>5555 5555
`AND A1,A0`	>5555 5555	>AAAA AAAA	xx1x	>0000 0000

Syntax **ANDI** *<IL>*,*<Rd>*

Execution IL AND (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	0	0	R	Rd			
~IL (LSW)															
~IL (MSW)															

Operands **IL** is a 32-bit immediate value.

Description ANDI bitwise-ANDs the value of the 32-bit immediate value, IL, with the contents of the destination register; the result is stored in the destination register.

This is an alternate mnemonic for `ANDNI IL,Rd`. The assembler stores the 1's complement of IL in the two extension words.

Words 3

Machine States 3,12

Status Bits
N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before A0	After NCZV	After A0
`ANDI >FFFFFFFF,A0`	>FFFF FFFF	xx0x	>FFFF FFFF
`ANDI >FFFFFFFF,A0`	>0000 0000	xx1x	>0000 0000
`ANDI >00000000,A0`	>0000 0000	xx1x	>0000 0000
`ANDI >AAAAAAAA,A0`	>5555 5555	xx1x	>0000 0000
`ANDI >AAAAAAAA,A0`	>AAAA AAAA	xx0x	> AAAA AAAA
`ANDI >55555555,A0`	>5555 5555	xx0x	>5555 5555
`ANDI >55555555,A0`	>AAAA AAAA	xx1x	>0000 0000

ANDN AND Register with Complement ANDN

Syntax **ANDN** *<Rs>,<Rd>*

Execution NOT(Rs) AND (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	Rs				R	Rd			

Description ANDN biwise-ANDs the 1's complement of the contents of the source register with the contents of the destination register; the result is stored in the destination register.

The source and destination registers must be in the same register file. Note that `ANDN Rn,Rn` has the same effect as `CLR Rn`.

Words 1

Machine States 1,4

Status Bits

N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before A1	Before A0	After NCZV	After A0
`ANDN A1,A0`	>FFFF FFFF	>FFFF FFFF	xx1x	>0000 0000
`ANDN A1,A0`	>FFFF FFFF	>0000 0000	xx1x	>0000 0000
`ANDN A1,A0`	>0000 0000	>0000 0000	xx1x	>0000 0000
`ANDN A1,A0`	>AAAA AAAA	>5555 5555	xx0x	>5555 5555
`ANDN A1,A0`	>AAAA AAAA	>AAAA AAAA	xx1x	>0000 0000
`ANDN A1,A0`	>5555 5555	>5555 5555	xx1x	>0000 0000
`ANDN A1,A0`	>5555 5555	>AAAA AAAA	xx0x	> AAAAAAAA

ANDNI — AND Not Immediate (32 Bits)

Syntax **ANDNI** *<IL>,<Rd>*

Execution NOT IL AND (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	0	0	R	Rd			
IL (LSW)															
IL (MSW)															

Operands **L** is a 32-bit immediate value.

Description ANDNI bitwise-ANDs the 1's complement of the 32-bit immediate data with the contents of the destination register; the result is stored in the destination register. ANDI also uses this opcode.

Words 3

Machine States 3,12

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** *1* if the result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before A0	After NCZV	After A0
`ANDNI >FFFFFFFF,A0`	>FFFF FFFF	xx1x	>0000 0000
`ANDNI >FFFFFFFF,A0`	>0000 0000	xx1x	>0000 0000
`ANDNI >00000000,A0`	>0000 0000	xx1x	>0000 0000
`ANDNI >AAAAAAAA,A0`	>5555 5555	xx0x	>5555 5555
`ANDNI >AAAAAAAA,A0`	>AAAA AAAA	xx1x	>0000 0000
`ANDNI >55555555,A0`	>5555 5555	xx1x	>0000 0000
`ANDNI >55555555,A0`	>AAAA AAAA	xx0x	>AAAA AAAA

BTST Test Register Bit - *Constant*

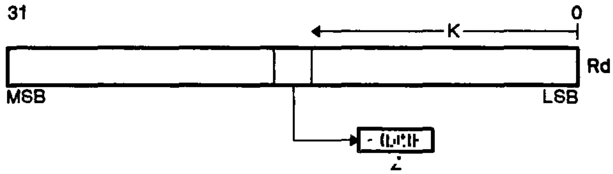
Syntax **BTST** *<K>,<Rd>*

Execution Set status on value of bit K in Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	~K					R	Rd			

Operands **K** is a constant in the range of 0 to 31.



Description BTST tests the specified destination register bit, K, and sets status bit Z accordingly. The K value must be an absolute expression that evaluates to a value in the range 0 to 31; if the value specified is greater than 31, the assembler issues a warning and truncates the K operand value to the five LSBs. The specified bit number is 1's complemented by the assembler before it is inserted into the K field of the opcode.

Words 1

Machine States 1,4

Status Bits
N Unaffected
C Unaffected
Z *1* if the bit tested is 0, *0* if the bit tested is 1.
V Unaffected

Examples

Code	Before	After
	A0	NCZV
BTST 0,A0	>5555 5555	xx0x
BTST 15,A0	>5555 5555	xx1x
BTST 31,A0	>5555 5555	xx1x
BTST 0,A0	>AAAA AAAA	xx1x
BTST 15,A0	>AAAA AAAA	xx0x
BTST 31,A0	>AAAA AAAA	xx0x
BTST 0,A0	>FFFF FFFF	xx0x
BTST 15,A0	>FFFF FFFF	xx0x
BTST 31,A0	>FFFF FFFF	xx0x
BTST 0,A0	>0000 0000	xx1x
BTST 15,A0	>0000 0000	xx1x
BTST 31,A0	>0000 0000	xx1x

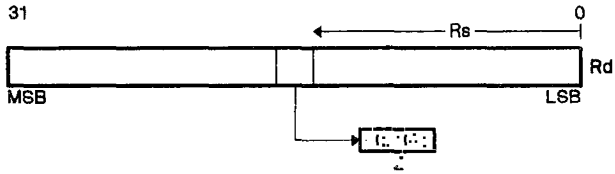
Syntax **BTST** *<Rs>,<Rd>*

Execution Set status on value of bit (Rs) in Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	Rs				R	Rd			

Operands **Rs** contains the number of the bit in Rd to be tested.



Description BTST tests the specified destination register bit and sets status bit Z accordingly. The five LSBs of the source register specify the bit to be tested (the 27 MSBs are ignored).

The source and destination registers must be in the same register file.

Words 1

Machine States 2,5

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** *1* if the bit tested is 0, *0* if the bit tested is 1.
- **V** Unaffected

Examples

Code	Before A1	Before A0	After NCZV
BTST A1,A0	>0000 0000	>5555 5555	xx0x
BTST A1,A0	>0000 000F	>5555 5555	xx1x
BTST A1,A0	>0000 001F	>5555 5555	xx1x
BTST A1,A0	>0000 0000	>AAAA AAAA	xx1x
BTST A1,A0	>0000 000F	>AAAA AAAA	xx0x
BTST A1,A0	>0000 001F	>AAAA AAAA	xx0x
BTST A1,A0	>FFFF FF8F	>FFFF 7FFF	xx0x
BTST A1,A0	>0000 0000	>FFFF FFFF	xx0x
BTST A1,A0	>0000 000F	>FFFF FFFF	xx0x
BTST A1,A0	>0000 001F	>FFFF FFFF	xx0x
BTST A1,A0	>0000 0000	>0000 0000	xx1x
BTST A1,A0	>0000 000F	>0000 0000	xx1x
BTST A1,A0	>0000 001F	>0000 0000	xx1x

Syntax **CALL** *<Rs>*

Execution

(PC') → TOS
(Rs) → PC
(SP) - 32 → SP

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	0	0	1	R	Rs			

Description

CALL pushes the address of the next instruction (PC') onto the stack, then jumps to a subroutine whose address is contained in the source register. This instruction can be used for indexed subroutine calls. Note that when Rs is the SP, Rs is decremented **after** being written to the PC (the PC contains the original value of Rs).

The TMS34010 always sets the four LSBs of the program counter to 0, so instructions are always word aligned.

The stack pointer (SP) points to the top of the stack; the stack is located in external memory. The stack grows in the direction of decreasing linear address. PC' is pushed onto the stack and the SP is predecremented by 32 before the return address is loaded onto the stack. Stack pointer alignment affects timing as indicated in **Machine States**, below.

Use the RETS instruction to return from a subroutine.

Words 1

Machine States

3+(3),9 (SP aligned)
3+(9),15 (SP nonaligned)

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Example

```
CALL A0
```

Before			After	
A0	**PC**	**SP**	**PC**	**SP**
>0123 4560	>0444 2210	>0F00 0020	>0123 4560	>0F00 0000

Memory will contain the following values after instruction execution:

Address	Data
>0F00 0010	>2220
>0F00 0020	>0444

Syntax **CALLA** *<Address>*

Execution (PC') → TOS
Address → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	1
Address (LSW)															
Address (MSW)															

Operands **Address** is a 32-bit absolute address.

Description CALLA pushes the address of the next instruction (PC') onto the stack, then jumps to the address contained in the two extension words. This instruction is used for long (greater than ±32K words) or externally referenced calls.

The lower four bits of the program counter are always set to 0, so instructions are always word-aligned.

The stack pointer (SP) points to the top of the stack; the stack is located in external memory. The stack grows in the direction of decreasing linear address. PC' is pushed onto the stack and the SP is predecremented by 32 before the return address is loaded onto the stack. Stack pointer alignment affects timing as indicated in **Machine States**, below.

Use the RETS instruction to return from a subroutine.

Words 3

Machine States 4+(2),15 (SP aligned)
4+(8),21 (SP nonaligned)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Example `CALLA >01234560`

Before		After	
PC	**SP**	**PC**	**SP**
>0444 2210	>0F00 0020	>0123 4560	>0F00 0000

Memory will contain the following values after instruction execution:

Address	Data
>0F00 0010	>2240
>0F00 0020	>0444

Syntax **CALLR** *<Address>*

Execution (PC') → TOS
PC' + (Displacement×16) → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1
Displacement															

Operands **Address** is a 32-bit address within ±32K words (-32,768 to 32,767) of PC'.

Description CALLR pushes the address of the next instruction (PC') onto the stack, then jumps to the subroutine at the address specified by the sum of the next instruction address and the signed word displacement. This instruction is used for calls within a specified module or section.

The displacement is computed by the assembler as (Address - PC')/16. The address must be defined within the section and within -32,768 to 32,767 words of the instruction following CALLR. The assembler will not accept an address value that is externally defined or defined within a different section than PC'.

The lower four bits of the program counter are always set to 0, so instructions are always word aligned.

The stack pointer (SP) points to the top of the stack; the stack is located in external memory. The stack grows in the direction of decreasing linear address. The PC is pushed on to the stack and the SP is predecremented by 32 before the return address is loaded onto the stack. Stack pointer alignment affects timing as indicated in **Machine States**, below.

Use the RETS instruction to return from a subroutine.

Words 2

Machine States 3+(2),11 (SP aligned)
3+(8),17 (SP nonaligned)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before PC	Before SP	After PC	After SP
CALLR >0447FFF0	>0440 0000	>0F00 0020	>0447 FFF0	>0F00 0000
CALLR >04480000	>0440 0000	>0F00 0020	>0448 0000	>0F00 0000

Memory will contain the following values after instruction execution:

Address	Data
>0F00 0010	>0000
>0F00 0020	>0440

Clear Register

Syntax **CLR** *<Rd>*

Execution (Rd) XOR (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	Rd				R	Rd			

Description CLR clears the destination register by XORing the contents of the register with itself. This is an alternate mnemonic for `XOR Rd,Rd`.

Words 1

Machine States 1,4

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** *1*
- **V** Unaffected

Examples

Code	Before	After	
	A0	A0	NCZV
`CLR A0`	>FFFF FFFF	>0000 0000	xx1x
`CLR A0`	>0000 0001	>0000 0000	xx1x
`CLR A0`	>8000 0000	>0000 0000	xx1x
`CLR A0`	>AAAA AAAA	>0000 0000	xx1x

Syntax CLRC

Execution 0 → C

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0

Description CLRC sets the status carry bit (C) to 0. The rest of the status register is unaffected. The SETC instruction is a counterpart to this instruction.

This instruction is useful for returning a true/false value (in the carry bit) from a subroutine without using a general-purpose register.

Words 1

Machine States 1,4

Status Bits

- **N** Unaffected
- **C** *0*
- **Z** Unaffected
- **V** Unaffected

Examples

Code	Before ST	Before NCZV	After ST	After NCZV
CLRC	>F000 0000	1111	>B000 0000	1011
CLRC	>4000 0010	0100	>0000 0010	0000
CLRC	>B000 001F	1011	>B000 001F	1011

Syntax **CMP** *<Rs>,<Rd>*

Execution Set status bits on the result of (Rd) - (Rs)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Rs				R	Rd			

Description CMP subtracts the contents of the source register from the contents of the destination register and sets the condition codes accordingly. Both the source and destination registers remain unaffected. This instruction is often used in conjunction with the JAcc or JRcc conditional jump instructions.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if a there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A1	A0	After NCZV	Jumps Taken
`CMP A1,A0`	>0000 0001	>0000 0001	0010	UC,NN,NC,Z,NV,LS,GE,LE,HS
`CMP A1,A0`	>0000 0001	>0000 0002	0000	UC,NN,NC,NZ,NV,P,HI,GE,GT,HS
`CMP A1,A0`	>0000 0001	>FFFF FFFF	1000	UC,N,NC,NZ,NV,P,HI,LT,LE,HS
`CMP A1,A0`	>0000 0001	>8000 0000	0001	UC,NN,NC,NZ,V,HI,LT,LE,HS
`CMP A1,A0`	>FFFF FFFF	>7FFF FFFF	1101	UC,N,C,NZ,V,LS,GE,GT,LO
`CMP A1,A0`	>FFFF FFFF	>8000 0000	1100	UC,N,C,NZ,NV,LS,LT,LE,LO
`CMP A1,A0`	>8000 0000	>7FFF FFFF	1101	UC,N,C,NZ,V,LS,GE,GT,LO

Syntax **CMPI** *<IW>*,*<Rd>* [,**W**]

Execution Set status bits on the result of (Rd) - IW

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	1	0	R	Rd			
~IW															

Operands **IW** is a 16-bit signed immediate value.

Description CMPI subtracts the sign-extended, 16-bit immediate data from the contents of the destination register and sets the condition codes accordingly. The destination register remains unaffected.

The assembler places the 1's complement of the specified value into the extension word (~IW).

The assembler will use the short form if the immediate value has been previously defined and is in the range -32,768 ≤ IW ≤ 32,767. You can force the assembler to use the short form by following the register specification with **W**:

```
CMPI  <IW>,<Rd>,W
```

The assembler will truncate the upper bits and issue an appropriate warning message if the value is greater than 16 bits.

This instruction is often used in conjunction with the JAcc or JRcc conditional jump instructions.

Words 2

Machine States 2,8

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A0	After NCZV	Jumps Taken
`CMPI 1,A0`	>0000 0002	0000	UC,NN,NC,NZ,NV,P,HI,GE,GT,HS
`CMPI 1,A0`	>0000 0001	0010	UC,NN,NC,Z,NV,LS,GE,LE,HS
`CMPI 1,A0`	>0000 0000	1100	UC,N,C,NZ,NV,LS,LT,LE,LO
`CMPI 1,A0`	>FFFF FFFF	1000	UC,N,NC,NZ,NV,P,HI,LT,LE,HS
`CMPI 1,A0`	>8000 0000	0001	UC,NN,NC,NZ,V,HI,LT,LE,HS
`CMPI -2,A0`	>0000 0000	0100	UC,NN,C,NZ,NV,P,LS,GE,GT,LO
`CMPI -2,A0`	>FFFF FFFF	0000	UC,NN,NC,NZ,NV,P,LI,GE,GT,HS
`CMPI -2,A0`	>FFFF FFFE	0010	UC,NN,NC,Z,NV,LS,GE,LE,HS
`CMPI -2,A0`	>FFFF FFFD	1100	UC,N,C,NZ,NV,LS,LT,LE,LO
`CMPI -1,A0`	>7FFF FFFF	1101	UC,N,C,NZ,V,LS,GE,GT,LO

CMPI — Compare Immediate - *32 Bits* — CMPI

Syntax **CMPI** *<IL>*,*<Rd>*[,**L**]

Execution Set status bits on the result of (Rd) - IL

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	1	1	R	Rd			
~IL (LSW)															
~IL (MSW)															

Operands **IL** is a 32-bit immediate value.

Description CMPI subtracts the signed, 32-bit immediate data from the contents of the destination register and sets the condition codes accordingly. The destination register remains unaffected.

The assembler places the 1's complement of the specified value into the extension words (~IL).

The assembler will use this opcode if it cannot use the short form. You can force the assembler to use the long form by following the register specification with **L**:

```
CMPI  <IL>,<Rd>,L
```

This instruction is often used in conjunction with the JAcc or JRcc conditional jump instructions.

Words 3

Machine States 3,12

Status Bits

- **N** *1* if the result is negative, *0* otherwise.
- **C** *1* if there is a borrow, *0* otherwise.
- **Z** *1* if the result is 0, *0* otherwise.
- **V** *1* if there is an overflow, *0* otherwise.

Examples

Code		Before A0	After NCZ	After V	Jumps Taken
CMPI	>8000,A0	>0000 8001	000	0	UC,NN,NC,NZ,NV,P,HI,GE,GT,HS
CMPI	>8000,A0	>0000 8000	001	0	UC,NN,NC,Z,NV,LS,GE,LE,HS
CMPI	>8000,A0	>0000 7FFF	110	0	UC,N,C,NZ,NV,LS,LT,LE,LO
CMPI	>8000,A0	>FFFF FFFF	100	0	UC,N,NC,NZ,NV,P,HI,LT,LE,HS
CMPI	>8000,A0	>8000 7FFF	000	1	UC,NN,NC,NZ,V,HI,LT,LE,HS
CMPI	>FFFF7FFF,A0	>0000 0000	010	0	UC,NN,C,NZ,NV,P,LS,GE,GT,LO
CMPI	>FFFF7FFE,A0	>FFFF 7FFF	000	0	UC,NN,NC,NZ,NV,P,HI,GE,GT,HS
CMPI	>FFFF7FFE,A0	>FFFF 7FFE	001	0	UC,NN,NC,Z,NV,LS,GE,LE,HS
CMPI	>FFFF7FFE,A0	>FFFF 7FFD	110	0	UC,N,C,NZ,NV,LS,LT,LE,LO
CMPI	>FFFF7FFF,A0	>7FFF 7FFF	110	1	UC,N,C,NZ,V,LS,GE,GT,LO

Syntax **CMPXY** *<Rs>,<Rd>*

Execution Set status bits on the results of:

(Rd**X**) - (Rs**X**)

(Rd**Y**) - (Rs**Y**)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	Rs				R	Rd			

Description CMPXY compares the source register to the destination register in XY mode and sets the status bits as if a subtraction had been performed. The registers themselves remain unaffected. The source and destination registers are treated as signed XY registers. Note that no overflow detection is provided.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

N *1* if source X field = destination X field, *0* otherwise.
C Sign bit of Y half of the result.
Z *1* if source Y field = destination Y field, *0* otherwise.
V Sign bit of X half of the result.

Examples

Code	Before A1	Before A0	After NCZV	Jumps Taken
`CMPXY A1,A0`	>0009 0009	>0001 0001	0101	NN,C,NZ,V,LS,LT
`CMPXY A1,A0`	>0009 0009	>0009 0001	0011	NN,NC,Z,V,LS,LT
`CMPXY A1,A0`	>0009 0009	>0001 0009	1100	N,C,NZ,NV,LS,LT
`CMPXY A1,A0`	>0009 0009	>0009 0009	1010	N,NC,Z,NV,LS,LT
`CMPXY A1,A0`	>0009 0009	>0000 0010	0100	NN,C,NZ,NV,LS,GE
`CMPXY A1,A0`	>0009 0009	>0009 0010	0010	NN,NC,Z,NV,LS,GE
`CMPXY A1,A0`	>0009 0009	>0010 0000	0001	NN,NC,NZ,V,HI,LT
`CMPXY A1,A0`	>0009 0009	>0010 0009	1000	N,NC,NZ,NV,HI,LT
`CMPXY A1,A0`	>0009 0009	>0010 0010	0000	NN,NC,NZ,NV,HI,GE

Compare Point to Window

Syntax **CPW** *<Rs>,<Rd>*

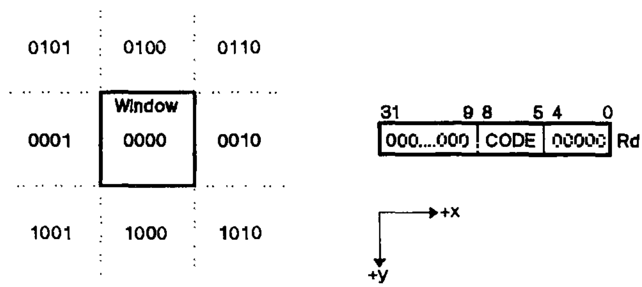
Execution Point Code → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	Rs				R	Rd			

Description CPW compares a point represented by an XY value in the source register to the window limits in the WSTART and WEND registers. The contents of the source register are treated as an XY address that consists of 16-bit signed X and Y values. WSTART and WEND are also treated as signed XY-format registers. WSTART and WEND should contain positive values; negative values produce unpredictable results. The location of the point with respect to the window is encoded as follows and loaded into the destination register.

Codes:



Note that the five LSBs of the destination register are set to 0 so that Rd can be used as an index into a table of 32-bit addresses.

This instruction can also be used to trivially reject lines that do not intersect with a window. The CPW codes for the two points defining the line are ANDed together. If the result is nonzero, then the line must lie completely outside the window (and does not intersect it). A 0 result indicates that the line *may* intersect the window, and a more rigorous test must be applied.

The source and destination registers must be in the same register file.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B5	WSTART	XY	Window start. Defines inclusive starting corner of window (lesser value corner).
B6	WEND	XY	Window end. Defines inclusive ending corner of window (greater value corner).

Words 1

Machine States 1,4

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** *1* if point lies outside window, *0* otherwise.

Examples

You must select appropriate implied operand values before executing the instruction. In this example, the implied operands are set up as follows, specifying a block of 36 pixels.

WSTART = 5,5
WEND = A,A

```
CPW A1,A0
```

Before		After	
A1	**NCZV**	**A0**	**NCZV**
>0004 0004	xxx0	>0000 00A0	xxx1
>0004 0005	xxx0	>0000 0080	xxx1
>0004 000A	xxx0	>0000 0080	xxx1
>0004 000B	xxx1	>0000 00C0	xxx1
>0005 0004	xxx1	>0000 0020	xxx1
>0005 0005	xxx0	>0000 0000	xxx0
>0005 000A	xxx0	>0000 0000	xxx0
>0005 000B	xxx0	>0000 0040	xxx1
>000A 0004	xxx0	>0000 0020	xxx1
>000A 0005	xxx1	>0000 0000	xxx0
>000A 000A	xxx1	>0000 0000	xxx0
>000A 000B	xxx0	>0000 0040	xxx1
>000B 0004	xxx0	>0000 0120	xxx1
>000B 0005	xxx0	>0000 0100	xxx1
>000B 000A	xxx0	>0000 0100	xxx1
>000B 000B	xxx0	>0000 0140	xxx1

Syntax **CVXYL** *<Rs>*,*<Rd>*

Execution (Rs **XY**) → Rd (Linear)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	Rs				R	Rd			

Operands **Rs** The source register contents are treated as an XY address that contains signed 16-bit X and Y values. The X value must be positive.

Description CVXYL converts an XY address to a linear address. The source register contains an XY address. The X value occupies the 16 LSBs of the register and the Y value occupies the 16 MSBs. This is converted into a 32-bit linear address which is stored in the destination register. The following conversion formula is used:

Address = (Y × Display Pitch) OR (X × Pixel Size) + Offset

Since the TMS34010 restricts the screen pitch and pixel size to powers of two (for XY addressing), the multiply operations in this conversion are actually shifts. The offset value is in the OFFSET register. The CONVDP value is used to determine the shift amount for the Y value, while the PSIZE register determines the X shift amount.

The source and destination registers must be in the same register file.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B3	DPTCH	Linear	Destination pitch
B4	OFFSET	Linear	Screen origin (location 0,0)
I/O Registers			
Address	**Name**	**Description and Elements (Bits)**	
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)	
>C0000150	PSIZE	Pixel size (1,2,4,8,16)	

Words 1

Machine States 3,6

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before	After			
	A0	OFFSET	PSIZE	CONVDP	A1
CVXYL A0,A1	>0040 0030	>0000 0000	>0010	>0014	>0002 0300
CVXYL A0,A1	>0040 0030	>0000 0000	>0008	>0014	>0002 0180
CVXYL A0,A1	>0040 0030	>0000 0000	>0004	>0014	>0002 0000
CVXYL A0,A1	>0040 0030	>0000 8000	>0004	>0014	>0002 8000
CVXYL A0,A1	>0040 0030	>0F00 0000	>0004	>0014	>0F02 0000
CVXYL A0,A1	>0040 0030	>0000 0000	>0002	>0014	>0002 0060
CVXYL A0,A1	>0040 0030	>0000 0000	>0001	>0014	>0002 0030
CVXYL A0,A1	>0040 0030	>0000 0000	>0001	>0013	>0004 0030
CVXYL A0,A1	>0040 0030	>0000 0000	>0001	>0015	>0001 0000

CONVDP = >0013 corresponds to DPTCH = >0000 1000
CONVDP = >0014 corresponds to DPTCH = >0000 0800
CONVDP = >0015 corresponds to DPTCH = >0000 0400

Syntax DEC <*Rd*>

Execution (Rd) - 1 → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	0	1	R	Rd			

Description DEC subtracts 1 from the contents of the destination register; the result is stored in the destination register. This instruction is an alternate mnemonic for `SUBK 1,Rd`.

Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the SUBB instruction.

Words 1

Machine States 1,4

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A1	After A1	NCZV
`DEC A1`	>0000 0010	>0000 000F	0000
`DEC A1`	>0000 0001	>0000 0000	0010
`DEC A1`	>0000 0000	>FFFF FFFF	1100
`DEC A1`	>FFFF FFFF	>FFFF FFFE	1000
`DEC A1`	>8000 0000	>7FFF FFFF	0001

Syntax DINT

Execution 0 → IE

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0

Description DINT disables interrupts by setting the global interrupt enable bit (IE, status bit 21) to 0. All interrupts except reset and NMI are disabled; the interrupt enable mask in the INTENB register is ignored. The remainder of the status register is unaffected.

The EINT instruction enables interrupts.

Words 1

Machine States 3,6

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected
- **IE** *0*

Examples

Code	Before ST	After ST
`DINT`	>0000 0010	>0000 0010
`DINT`	>0020 0010	>0000 0010

Syntax **DIVS** *<Rs>,<Rd>*

Execution Rd Even: (Rd):(Rd+1)/(Rs) → Rd, remainder → Rd+1
Rd Odd: (Rd)/(Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rs				R	Rd			

Operands

Rs is a 32-bit signed divisor.

Rd is a 32-bit signed dividend, or the most significant half of a 64-bit signed dividend.

Description There are two cases:

Rd Even DIVS performs a signed divide of the 64-bit operand contained in the two consecutive registers, starting at the specified destination register, by the 32-bit contents of the source register. The specified even-numbered destination register, Rd, contains the 32 MSBs of the dividend. The next consecutive register (which is odd-numbered) contains the 32 LSBs of the dividend. The quotient is stored in the destination register, and the remainder is stored in the following register (Rd+1). The remainder is always the same sign as the dividend (in Rd:Rd+1). Avoid using A14 or B14 as the destination register, since this overwrites the SP; the assembler will issue a warning in this case.

Rd Odd DIVS performs a signed divide of the 32-bit operand contained in the destination register by the 32-bit value in the source register. The quotient is stored in the destination register; the remainder is not returned.

The source and destination registers must be in the same register file.

Words 1

Machine States
40,43 (Rd even)
39,42 (Rd odd)
41,44 if result = >80000000
7,10 if (Rd) $\geq$ (Rs) or (Rs) $\leq$ 0

Status Bits
N *1* if the quotient is negative, *0* otherwise.
C Unaffected
Z *1* if the quotient is 0, *0* otherwise.
V *1* if quotient overflows (cannot be represented by 32 bits), *0* otherwise. The following conditions will set the overflow flag:

- Divisor is 0
- Quotient cannot be contained within 32 bits

Examples

```
DIVS A2,A0
```

Before A0	Before A1	Before A2	After A0	After A1	After A2	NCZV
>1234 5678	>8765 4321	>8765 4321	>D95B C60A	>15CA 1DD7	>8765 4321	1x00
>EDCB A987	>789A BCDF	>8765 4321	>26A4 39F6	>EA35 E229	>8765 4321	0x00
>EDCB A987	>789A BCDF	>789A BCDF	>D95B C60A	>EA35 E229	>789A BCDF	1x00
>1234 5678	>8765 4321	>789A BCDF	>26A4 39F6	>15CA 1DD7	>789A BCDF	0x00
>1234 5678	>8765 4321	>0000 0000	>1234 5678	>8765 4321	>0000 0000	0x01
>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	0x01
>0000 0000	>0000 0000	>8765 4321	>0000 0000	>0000 0000	>8765 4321	0x10
>8765 4321	>0000 0000	>8765 4321	>8765 4321	>0000 0000	>8765 4321	0x01

```
DIVS A2,A1
```

Before A0	Before A1	Before A2	After A0	After A1	After A2	NCZV
>0000 0000	>8765 4321	>1234 5678	>0000 0000	>FFFF FFFA	>1234 5678	1x00
>0000 0000	>8765 4321	>EDCB A988	>0000 0000	>0000 0006	>EDCB A988	0x00
>0000 0000	>789A BCDF	>EDCB A988	>0000 0000	>FFFF FFFA	>EDCB A988	1x00
>0000 0000	>789A BCDF	>1234 5678	>0000 0000	>0000 0006	>1234 5678	0x00
>0000 0000	>8765 4321	>0000 0000	>0000 0000	>8765 4321	>0000 0000	0x01
>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	0x01

Syntax DIVU *<Rs>,<Rd>*

Execution Rd Even: (Rd):(Rd+1)/(Rs) → Rd, remainder → Rd+1
Rd Odd: (Rd)/(Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	Rs				R	Rd			

Operands

Rs is a 32-bit unsigned divisor.

Rd is a 32-bit unsigned dividend or the most significant half of a 64-bit unsigned divisor.

Description There are two cases:

Rd Even DIVU performs an unsigned divide of the 64-bit operand contained in the two consecutive registers, starting at the destination register, by the 32-bit contents of the source register. The specified even-numbered destination register, Rd, contains the 32 MSBs of the dividend. The next consecutive register (which is odd-numbered) contains the 32 LSBs of the dividend. The quotient is stored in the destination register, and the remainder is stored in the following register (Rd+1). Avoid using A14 or B14 as the destination register, since this overwrites the SP; the assembler will issue a warning in this case.

Rd Odd DIVU performs an unsigned divide of the 32-bit operand contained in the destination register by the 32-bit value in the source register. The quotient is stored in the destination register; the remainder is not returned.

The source and destination registers must be in the same register file.

Words 1

Machine States

37,40 (Rd even)
37,40 (Rd odd)
5,8 if (Rd) $\geq$ (Rs) or (Rs) $\leq$ 0

Status Bits

N Unaffected
C Unaffected
Z *1* if the quotient is 0, *0* otherwise.
V *1* if quotient overflows (cannot be represented by 32 bits), *0* otherwise. The following conditions set the overflow flag:

- Divisor is 0
- Quotient cannot be contained within 32 bits

Examples

```
DIVU A2,A0
```

Before A0	Before A1	Before A2	After A0	After A1	After A2	NCZV
>1234 5678	>8765 4321	>789A BCDF	>26A4 39F6	>15CA 1DD7	>789A BCDF	xx00
>1234 5678	>8765 4321	>0000 0000	>1234 5678	>8765 4321	>0000 0000	xx01
>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	xx01
>0000 0000	>0000 0000	>8765 4321	>0000 0000	>0000 0000	>8765 4321	xx10
>8765 4321	>0000 0000	>8765 4321	>8765 4321	>0000 0000	>8765 4321	xx01

```
DIVU A2,A1
```

Before A0	Before A1	Before A2	After A0	After A1	After A2	NCZV
>0000 0000	>789A BCDF	>1234 5678	>0000 0000	>0000 0006	>1234 5678	xx00
>0000 0000	>1234 5678	>0000 0000	>0000 0000	>1234 5678	>0000 0000	xx01
>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	>0000 0000	xx01
>0000 0000	>0000 0000	>8765 4321	>0000 0000	>0000 0000	>8765 4321	xx10
>0000 0000	>8765 4321	>8765 4321	>0000 0000	>0000 0001	>8765 4321	xx00

Syntax **DRAV** *<Rs>,<Rd>*

Execution

(pixel)COLOR1 → *Rd
(Rs**X**) + (Rd**X**) → Rd**X**
(Rs**Y**) + (Rd**Y**) → Rd**Y**

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	Rs				R	Rd			

Description DRAV writes the pixel value in the COLOR1 register to the location pointed to by the XY address in the destination register. Following the write, the XY address in the destination register is incremented by the value in the source register: the X half of Rs is added to the X half of Rd, and the Y half of Rs is added to the Y half of Rd. Any carry out from the lower (X) half of the register will not propagate into the upper (Y) half.

COLOR1 bits 0–15 are output on data bus lines 0–15, respectively. The pixel data used from COLOR1 is that which aligns to the destination location, so 16-bit patterns can be implemented. The source and destination registers must be in the same register file.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B3	DPTCH	Linear	Destination pitch
B4	OFFSET	Linear	Screen origin (location 0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner
B9	COLOR1	Pixel	Pixel color
I/O Registers			
Address	**Name**	**Description and Elements (Bits)**	
>C00000B0	CONTROL	PP – Pixel processing operations (22 options) W – Window checking operation T – Transparency operation	
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)	
>C0000150	PSIZE	Pixel size (1,2,4,8,16)	
>C0000160	PMASK	Plane mask – pixel format	

Pixel Processing Set the PPOP field in the CONTROL register to select a pixel processing operation. This operation will be applied to the pixel as it is moved to the destination location. At reset, the default pixel processing operation is *replace* (S → D). For more information, see Section 7.7, Pixel Processing, on page 7-15.

Window Checking Select a window checking mode by setting the W bits in the CONTROL register. If you select an active window checking mode (W = 1, 2, or 3), the WSTART and WEND registers will define the XY starting and ending corners of a rectangular window. The X and Y values in both WSTART and WEND must be positive.

When the TMS34010 attempts to write a pixel inside or outside a defined value, the following actions may occur:

W=0 No window operation. The pixel is drawn and the WVP and V bits are unaffected.

W=1 Window hit. No pixels are drawn. The V bit is set to 0 if the pixel lies within the window; otherwise, it is set to 1.

W=2 Window miss. If the pixel lies outside the window, the WVP and V bits are set to 1 and the instruction is aborted (no pixels are drawn). Otherwise, the pixel is drawn and the V bit is set to 0.

W=3 Window clip. If the pixel lies outside the window, the V bit is set to 1 and the instruction is aborted (no pixels are drawn). Otherwise, the pixel is drawn and the V bit is set to 0.

For more information, see Section 7.10, Window Checking, on page 7-25.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL register to 1. The TMS34010 checks for 0-valued (transparent) pixels resulting from the combination of the source and destination pixels, according to the selected pixel processing operation. At reset, the default case for transparency is *off*.

Plane Mask The plane mask is enabled for this instruction.

Shift Register Transfers When this instruction is executed and the SRT bit is set, normal memory read and write operations become SRT reads and writes. Refer to Section 9.9.2, Video Memory Bulk Initialization, on page 9-27 for more information.

Words 1

Machine States The states consumed depend on the operation selected, as indicated below.

Pixel Processing Operation								Window Violation		
PSIZE	**Replace**	**Boolean**	**ADD**	**ADDS**	**SUB**	**SUBS**	**MIN/MAX**	**W=1**	**W=2**	**W=3**
1,2,4,8	4+(3),10	6+(3),12	7+(3),13	7+(3),13	7+(3),13	8+(3),14	7+(3),13	5,8	3,6	5,8
16	4+(1),8	6+(1),10	6+(1),10	7+(1),11	7+(1),11	8+(1),12	7+(1),11	5,8	3,6	5,8

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** *1* if a window violation occurs, *0* otherwise; unaffected if window clipping is not used.

Examples These DRAV examples use the following implied operand setup.

Register File B:
DPTCH (B3) = >200
OFFSET (B4) = >0001 0000
WSTART (B5) = >0010 0000
WEND (B6) = >003C 0040
COLOR1 (B9) = >FFFF FFFF

I/O Registers:
CONVDP = >0016

Assume that memory contains the following values before instruction execution:

Address	Data
>0001 8040	>8888

Code	Before					After		
	A0	**A1**	**PSIZE**	**PP**	**W**	**PMASK**	**A0**	**@>18040**
DRAV A1,A0	>0040 0040	>0010 0010	>0001	00000	00	>0000	>0050 0050	>8889
DRAV A1,A0	>0040 0020	>0010 0010	>0002	00000	00	>0000	>0050 0030	>888B
DRAV A1,A0	>0040 0010	>0010 0010	>0004	00000	00	>0000	>0050 0020	>888F
DRAV A1,A0	>0040 0008	>0010 0010	>0008	00000	00	>0000	>0050 0018	>88FF
DRAV A1,A0	>0040 0004	>0010 0010	>0010	00000	00	>0000	>0050 0014	>FFFF
DRAV A1,A0	>0040 0004	>0000 FFFF	>0010	01010	00	>0000	>0040 0003	>0000
DRAV A1,A0	>0040 0004	>FFFF 0000	>0010	10011	00	>0000	>003F 0004	>0000
DRAV A1,A0	>0040 0004	>0001 0001	>0010	00000	11	>0000	>0041 0005	>0000
DRAV A1,A0	>0040 0004	>0040 0004	>0010	00000	00	>00FF	>0080 0008	>FF00

DSJ — Decrement Register and Skip Jump

Syntax **DSJ** *<Rd>,<Address>*

Execution
(Rd) - 1 → Rd
If (Rd) ≠ 0, then (Displacement×16) + (PC') → PC
If (Rd) = 0, then go to next instruction

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	0	0	R	Rd			
Displacement															

Operands

Rd contains the operand to be decremented.

Address is a 32-bit address (within 32K words).

Description

DSJ decrements the contents of the destination register by 1. If this result is **nonzero**, then a jump is made relative to the current PC. The current PC points to the instruction word that immediately follows the second word of the DSJ instruction. The signed word displacement is converted to a bit displacement by multiplying by 16. The new PC address is then obtained by adding the resulting signed displacement (Displacement × 16) to the address of the next instruction.

If the result of the destination register decrement is **0**, then no jump is performed and the program continues execution at the next sequential instruction.

The displacement is computed by the assembler as (Address - PC')/16. The resulting jump range is -32,768 to +32,767 words. The specified 32-bit address is converted by the assembler into the value required for the displacement field.

This instruction is useful for large loops involving a counter. For shorter loops, the assembler will translate this into a DSJS instruction.

Words 2

Machine States
3,9 (Jump)
2,8 (No jump)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before A5	After A5	Jump taken?
`DSJ A5,LOOP`	>0000 0009	>0000 0008	Yes
`DSJ A5,LOOP`	>0000 0001	>0000 0000	No
`DSJ A5,LOOP`	>0000 0000	>FFFF FFFF	Yes

Conditionally Decrement Register and Skip Jump

Syntax **DSJEQ** *<Rd>,<Address>*

Execution

If (Z) = 1 then (Rd) - 1 → Rd
  If (Rd) ≠ 0 then PC′ + (Displacement×16) → PC
  If (Rd) = 0 then go to next instruction
If (Z) = 0 then go to next instruction

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	0	1	R	Rd			
Displacement															

Operands

Rd contains the operand to be conditionally decremented.

Address is a 32-bit address (within 32K words).

Description The DSJEQ instruction performs a conditional jump, based on an evaluation of the status Z bit.

- If **Z = 1**, the contents of the destination register are decremented by 1.
  - If this result is **nonzero**, then a jump is made relative to the current PC. The current PC points to the instruction word that immediately follows the second word of the DSJ instruction. The signed word displacement is converted to a bit displacement by multiplying by 16. The new PC address is then obtained by adding the resulting signed displacement (Displacement × 16) to the address of the next instruction.
  - If the result is **0**, then the jump is skipped and the program continues execution at the next sequential instruction.
- If **Z = 0**, the jump is skipped, the program counter is advanced to the next sequential instruction, and the instruction completes.

The displacement is computed by the assembler as (Address - PC′)/16. The resulting jump range is -32,768 to +32,767 words. The specified 32-bit address is converted by the assembler into the value required for the displacement field.

This instruction can be used after an explicit or implicit compare to 0. Additional information on these types of compares can be obtained in the CMP and CMPI, and MOVE-to-register instructions, respectively.

Words 2

Machine States

3,9 (Jump)
2,8 (No jump)

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before		After	
	A5	NCZV	A5	Jump taken?
DSJEQ A5,LOOP	>0000 0009	xx1x	>0000 0008	Yes
DSJEQ A5,LOOP	>0000 0001	xx1x	>0000 0000	No
DSJEQ A5,LOOP	>0000 0000	xx1x	>FFFF FFFF	Yes
DSJEQ A5,LOOP	>0000 0009	xx0x	>0000 0009	No
DSJEQ A5,LOOP	>0000 0001	xx0x	>0000 0001	No
DSJEQ A5,LOOP	>0000 0000	xx0x	>0000 0000	No

Conditionally Decrement Register and Skip Jump

Syntax **DSJNE** *<Rd>,<Address>*

Execution
If (Z) = 0 then (Rd) - 1 → Rd
    If (Rd) ≠ 0 then PC' + (Displacement×16) → PC
    If (Rd) = 0 then go to next instruction
If (Z) = 1 then to to next instruction

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	1	0	R	Rd			
Displacement															

Operands

Rd contains the operand to be conditionally decremented.

Address is a 32-bit address (within 32K words).

Description The DSJNE instruction performs a conditional jump, based on an evaluation of the Z bit.

- If **Z = 0**, the contents of the destination register are decremented by 1.
  - If this result is **nonzero**, then a jump is made relative to the current PC. The current PC points to the instruction word that immediately follows the second word of the DSJ instruction. The signed word displacement is converted to a bit displacement by multiplying by 16. The new PC address is then obtained by adding the resulting signed displacement (Displacement × 16) to the address of the next instruction.
  - If the result is **0**, then the jump is skipped and the program continues execution at the next sequential instruction.
- If **Z = 1**, the jump is skipped, the program counter is advanced to the next sequential instruction, and the instruction completes.

The displacement is computed by the assembler as (Address - PC')/16. The resulting jump range is -32,768 to +32,767 words. The specified 32-bit address is converted by the assembler into the value required for the displacement field.

This instruction can be used after an explicit compare or an implicit compare to 0. Additional information on these types of compares can be obtained in the CMP, CMPI, and MOVE-to-register instructions.

Words 2

Machine States
3,9 (Jump)
2,8 (No jump)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before A5	NCZV	After A5	Jump taken?
DSJNE A5,LOOP	>0000 0009	xx1x	>0000 0009	No
DSJNE A5,LOOP	>0000 0001	xx1x	>0000 0001	No
DSJNE A5,LOOP	>0000 0000	xx1x	>0000 0000	No
DSJNE A5,LOOP	>0000 0009	xx0x	>0000 0008	Yes
DSJNE A5,LOOP	>0000 0001	xx0x	>0000 0000	No
DSJNE A5,LOOP	>0000 0000	xx0x	>FFFF FFFF	Yes

Syntax **DSJS** *<Rd>,<Address>*

Execution
(Rd) - 1 → Rd
If (Rd) ≠ 0 then PC' + (Displacement×16) → PC
If (Rd) = 0 then go to next instruction

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	D	Displacement					R	Rd			

Operands

Rd contains the operand to be decremented.

Address is a 32-bit address (within 32K words).

Description

DSJS performs a conditional jump; first, it decrements the contents of the destination register by 1.

- If this result is **nonzero**, then a jump is made relative to the current PC. The current PC points to the instruction word that immediately follows the second word of the DSJ instruction. The 5-bit displacement is converted to a bit displacement by multiplying by 16.
  - If the direction bit D is **0**, the new PC address is then obtained by adding the resulting displacement to PC'.
  - If the direction bit D is **1**, the new PC address is obtained by subtracting the resulting displacement from PC'. This provides a jump range of -32 to 32 words, excluding 0.
- If the result of the decrement is **0**, then the jump is skipped and program execution continues at the next sequential instruction.

The specified 32-bit address is converted by the assembler into the value required for the displacement field. The displacement is computed by the assembler as (Address - PC')/16. This instruction is useful for coding tight loops for cache-resident routines.

Words 1

Machine States
2,5 (Jump)
3,6 (No jump)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before A5	After A5	Jump taken?
`DSJS A5,LOOP`	>0000 0009	>0000 0008	Yes
`DSJS A5,LOOP`	>0000 0001	>0000 0000	No
`DSJS A5,LOOP`	>0000 0000	>FFFF FFFF	Yes

Syntax	**EINT**
Execution	1 → IE

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0

Description EINT sets the global interrupt enable bit (IE) to 1, allowing interrupts to be enabled. When IE=1, individual interrupts can be enabled by setting the appropriate bits in the INTENB interrupt mask register. The rest of the status register is unaffected.

The DINT instruction disables interrupts.

Words 1

Machine States 3,6

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected
IE *1*

Examples

Code	Before	After
	ST	ST
`EINT`	>00000010	>00200010
`EINT`	>00200010	>00200010

Syntax EMU

Execution ST → Rd and conditionally enter emulator mode

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Description The EMU instruction pulses the $\overline{\text{EMUA}}$ pin and samples the RUN/$\overline{\text{EMU}}$ pin. If the RUN/$\overline{\text{EMU}}$ pin is in the RUN state, the EMU instruction acts as a NOP. If the pin is in the EMU state, emulation mode is entered. This instruction is not intended for general use; refer to the *TMS34010 XDS/22 User's Guide* for more information.

Words 1

Machine States 6,9 (or more if EMU mode is entered)

Status Bits
- **N** Indeterminate
- **C** Indeterminate
- **Z** Indeterminate
- **V** Indeterminate

Syntax **EXGF** *<Rd>*[,*<F>*]

Execution (Rd) → FS0, FE0 **or** (Rd) → FS1, FE1
FS0, FE0 → (Rd) **or** FS1, FE1 → (Rd)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	F	1	0	0	0	Rd				

Operands F is an optional operand; it defaults to 0.
F=0 selects FS0, FE0 to be exchanged.
F=1 selects FS1, FE1 to be exchanged

Description EXGF exchanges the six LSBs of the destination register with the selected six bits of field information (field size and field extension). Bit 5 of the 6-bit quantity in Rd is exchanged with the field extension value. The upper 26 bits of Rd are cleared.



Status Register

Words 1

Machine States 1,4

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before A5	Before ST	After A5	After ST
`EXGF A5,0`	>FFFF FFC0	>F000 0FFF	>0000 003F	>F000 0FC0
`EXGF A5,1`	>FFFF FFC0	>F000 0FFF	>0000 003F	>F000 003F

Syntax EXGPC *<Rd>*

Execution (Rd) → PC, (PC') → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	1	R	Rd			

Description EXGPC exchanges the next program counter value with the destination register contents. After this instruction has been executed, the destination register contains the address of the instruction immediately following the EXGPC instruction.

Note that the TMS34010 sets the four LSBs of the program counter to 0 (word aligned).

This instruction provides a "quick call" capability by saving the return address in a register (rather than on the stack). The return from the call is accomplished by repeating the instruction at the end of the "subroutine." Note that the subroutine address must be reloaded following each call-return operation.

Words 1

Machine States 2,5

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples

Code	Before A1	Before PC	After A1	After PC
`EXGPC A1`	>0000 1C10	>0000 2080	>0000 2090	>0000 1C10
`EXGPC A1`	>0000 1C50	>0000 2080	>0000 2090	>0000 1C50

Syntax **FILL L**

Execution pixel(COLOR1) → Pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0

Operands **L** specifies that the pixel array starting address is in linear format.

Description FILL processes a set of source pixel values (specified by the COLOR1 register) with a destination pixel array. This instruction operates on a two-dimensional array of pixels using pixels defined in the COLOR1 register. As the FILL proceeds, the source pixels are combined with destination pixels based on the selected graphics operations.

Note that the instruction is entered as `FILL L`. The following set of implied operands govern the operation of the instruction and define both the source pixels and the destination array.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B2†	DADDR	Linear	Pixel array starting address
B3	DPTCH	Linear	Pixel array pitch
B7	DYDX	XY	Pixel array dimensions (rows:columns)
B9	COLOR1	Pixel	Fill color or 16-bit pattern
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Operations**
>C00000B0	CONTROL	PP – Pixel processing operations (22 options) T – Transparency operation
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† Changed by FILL during execution.

Destination Array

The contents of the DADDR, DPTCH, and DYDX registers define the location of the destination pixel array:

- At the outset of the instruction, DADDR contains the **linear** address of the pixel with the lowest address in the array.

  During instruction execution, DADDR points to the next pixel (or word of pixels) to be modified in the destination array. When the array transfer is complete, DADDR points to the linear address of the pixel following the last pixel written.

- DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array. DPTCH must be a multiple of 16, exept when a single pixel-width line is drawn (DX=1). In this case, DPTCH may be any value.

- DYDX specifies the dimensions of the destination array in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns.

Pixel Processing Set the PPOP field in the CONTROL register to select a pixel processing operation. This operation will be applied to the pixel as it is moved to the destination location. There are 16 Boolean and 6 arithmetic operations; the default operation at reset is *replace* (S → D). Note that the destination data is read through the plane mask and then processed. The 6 arithmetic operations do not operate with pixel sizes of one or two bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Window Checking Window checking **cannot** be used with this instruction. The contents of the WSTART and WEND registers are ignored.

Corner Adjust There is no corner adjust for this instruction. The direction of the FILL is fixed as increasing linear addresses.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it processes the source data. At reset, the default case for transparency is *off*.

Interrupts This instruction can be interrupted at a word or row boundary of the destination array. When the FILL is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values. DADDR points to the linear address of the next word of pixels to be modified after the insterrupt is processed. SADDR points to the address of the next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the FILL correctly. You can inhibit the TMS34010 from resuming the FILL by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Plane Mask The plane mask is enabled for this instruction.

Shift Register Transfers If the SRT bit in the DPYCTL register is set, each memory read or write initiated by the FILL generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory clears or transfers. (Not all VRAMs support this capability.) See Section 9.9.2, Video Memory Bulk Initialization, on page 9-27 for more information.

Words 1

Machine States See Section 13.3, FILL Instructions Timing.

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples

These FILL examples use the following implied operand setup.

Register File B:		**I/O Registers:**	
DADDR (B2)	= >00002010	PSIZE	= >0008
DPTCH (B3)	= >00000080		
DYDX (B7)	= >0002000D		
COLOR1 (B9)	= >30303030		

Assume that memory contains the following values before instruction execution.

Linear Address	**Data**
>02000	>1100, >3322, >5544, >7766, >9988, >BBAA,>DDCC,>FFEE
>02080	>1100, >3322, >5544, >7766, >9988, >BBAA,>DDCC,>FFEE

Example 1

This example uses the pixel processing *replace (S → D)* operation. Before instruction execution, PMASK = >0000 and CONTROL = >0000 (T=0, PP=00000).

After instruction execution, memory contains the following values:

Linear Address	**Data**
>02000	>1100, >3030, >3030, >3030, >3030, >3030, >3030, >FF30
>02080	>1100, >3030, >3030, >3030, >3030, >3030, >3030, >FF30

Example 2

This example uses the *($\overline{S}$ and D) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >2C00 (T=0, PP=01010).

After instruction execution, memory contains the following values:

Linear Address	**Data**
>02000	>1100, >0302, >4544, >4746, >8988, >8B8A,>CDCC,>FFCE
>02080	>1100, >0302, >4544, >4746, >8988, >8B8A,>CDCC,>FFCE

Example 3

This example uses transparency and the *(S and D) → D* pixel processing operation. Before instruction execution, PMASK = > 0000 and CONTROL = > 0420 (T=1, PP=00000).

After instruction execution, memory contains the following values:

Linear Address	**Data**
>02000	>1100, >3020, >1044, >3020, >1088, >3020, >10CC,>FF20
>02080	>1100, >3020, >1044, >3020, >1088, >3020, >10CC,>FF20

Example 4 This example uses plane masking; the four MSBs are masked. Before instruction execution, PMASK = >F0F0 and CONTROL = >0000 (T=0, PP=00000).

After instruction execution, memory contains the following values:

Linear Address	Data
>02000	>1100, >3020, >5040, >7060, >9080, >B0A0, >D0C0, >FFE0
>02080	>1100, >3020, >5040, >7060, >9080, >B0A0, >D0C0, >FFE0

Syntax FILL XY

Execution pixel(COLOR1) → Destination pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0

Operands **XY** Specifies that the pixel array starting address is given in XY format.

Description FILL processes a set of source pixel values (specified by the COLOR1 register) with a destination pixel array.

This instruction operates on a two-dimensional array of pixels using pixels defined in the COLOR1 register. As the FILL proceeds, the source pixels are combined with destination pixels based on the selected graphics operations.

Note that the instruction is entered as `FILL L,XY`. The following set of implied operands govern the operation of the instruction and define both the source pixels and the destination array.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B2†‡	DADDR	XY	Pixel array starting address
B3	DPTCH	Linear	Pixel array pitch
B4	OFFSET	Linear	Screen origin (address of 0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner
B7†‡	DYDX	XY	Pixel array dimensions (rows:columns)
B9	COLOR1	Pixel	Fill color or 16-bit pattern
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP – Pixel processing operations (22 options) W – Window checking operation T – Transparency operation
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† Changed by FILL during execution.
‡ Used for common rectangle function with window hit operation (W=1).

Destination Array

The location of the destination pixel array is defined by the contents of the DADDR, DPTCH, CONVDP, OFFSET, and DYDX registers. At the outset of the instruction, DADDR contains the **XY** address of the pixel with the lowest address in the array. It is used with OFFSET and CONVDP to calculate the linear address of the starting location of the array. DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array (typically this is the screen pitch). DPTCH must be a power of two (greater than or equal to 16) and CONVDP must be set to

correspond to the DPTCH value. CONVDP is computed by operating on the DPTCH register with the LMO instruction; it is used for the XY calculations involved in XY addressing and window clipping. DYDX specifies the dimensions of the destination array in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns. During instruction execution, DADDR points to the next pixel (or word of pixels) to be modified in the destination array. When the array transfer is complete, DADDR points to the linear address of the pixel following the last pixel written. This is that pixel on the **last** row that would have been written had the array transfer been wider in the X dimension.

Pixel Processing

Pixel processing can be used with this instruction. The PPOP field of the CONTROL register specifies the pixel processing operation that will be applied to pixels as they are processed with the destination array. There are 16 Boolean and 6 arithmetic operations; the default case at reset is the *replace (S → D)* operation. Note that the destination data is read through the plane mask and then processed. The 6 arithmetic operations do not operate with pixel sizes of one or two bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Window Checking

The window operations described in Section 7.10, Window Checking, on page 7-25. can be used with this instruction. Window pick, violation detect, or preclipping can be selected by setting the W bits in the CONTROL register to 1, 2, or 3, respectively. Window pick modifies the DADDR and DYDX registers to correspond to the common rectangle formed by the destination array and the clipping window defined by WSTART and WEND. DADDR is set to the XY address of the pixel with the lowest address in the common rectangle, while DYDX is set to the X and Y dimensions of the rectangle. If no window operations are selected, the WSTART and WEND registers are ignored. At reset, no window operations are enabled.

Corner Adjust

There is no corner adjust for this instruction. The direction of the FILL is fixed as increasing linear addresses.

Transparency

Transparency can be enabled for this instruction by setting the T bit in the CONTROL register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it processes the source data. At reset, the default case for transparency is *off*.

Interrupts

This instruction can be interrupted at a word or row boundary of the destination array. When the FILL is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values. DADDR points to the linear address of the next word of pixels to be modified after the interrupt is processed. SADDR points to the address of the next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the FILL correctly. You can inhibit the TMS34010 from resuming the FILL by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Plane Mask The plane mask is enabled for this instruction.

Shift Register Transfers If the SRT bit in the DPYCTL register is set, each memory read or write initiated by the FILL generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory clears or transfers. (Not all VRAMs support this capability.) See Section 9.9.2, Video Memory Bulk Initialization, on page 9-27 for more information.

Words 1

Machine States See Section 13.3, FILL Instructions Timing.

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V *1* if a window violation occurs, *0* otherwise. Unaffected if window clipping is not enabled.

Examples These FILL examples use the following implied operand setup.

Register File B:		**I/O Registers:**	
DADDR (B2)	= >0052 0007	CONVDP	= >0017
DPTCH (B3)	= >0000 0100	PSIZE	= >0004
OFFSET (B4)	= >0001 0000	PMASK	= >0000
WSTART (B5)	= >0030 000C	CONTROL	= >0000
WEND (B6)	= >0053 0014		(W=00, T=0, PP=00000)
DYDX (B7)	= >0003 0012		
COLOR1 (B9)	= >FFFF FFFF		

Assume that memory contains the following values before instruction execution.

Linear Address	Data
>15200	>3210, >7654, >BA98,>FEDC,>3210, >7654, >BA98,>FEDC
>15300	>3210, >7654, >BA98,>FEDC,>3210, >7654, >BA98,>FEDC
>15400	>3210, >7654, >BA98,>FEDC,>3210, >7654, >BA98,>FEDC

Example 1 This example uses the *replace* (S → D) pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory contains the following values:

Linear Address	Data
>15200	>3210, >F654, >FFFF, >FFFF, >FFFF, >FFFF, >BA9F,>FEDC
>15300	>3210, >F654, >FFFF, >FFFF, >FFFF, >FFFF, >BA9F,>FEDC
>15400	>3210, >F654, >FFFF, >FFFF, >FFFF, >FFFF, >BA9F,>FEDC

XY Addressing

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	0	1	2	3	4	5	6	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	9	A	B	C	D	E	F
53	0	1	2	3	4	5	6	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	9	A	B	C	D	E	F
54	0	1	2	3	4	5	6	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	9	A	B	C	D	E	F

Example 2 This example uses the *(D XOR S) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >2800 (T=0, W=00, PP=01010).

After instruction execution, memory contains the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	0	1	2	3	4	5	6	8	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8	7	9	A	B	C	D	E	F
53	0	1	2	3	4	5	6	8	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8	7	9	A	B	C	D	E	F
54	0	1	2	3	4	5	6	8	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8	7	9	A	B	C	D	E	F

Example 3 This example uses transparency, the *(D subs S) → D* pixel processing operation. Before instruction execution, COLOR1 = >88888888, PMASK = >0000, and CONTROL = >4C20 (T=1, W=00, PP=10011).

After instruction execution, memory contains the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	0	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
53	0	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
54	0	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Example 4

This example uses window operation 3; the destination is clipped. Before instruction execution, PMASK = >0000 and CONTROL = >00C0 (T=0, W=11, PP=00000).

After instruction execution, memory contains the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	0	1	2	3	4	5	6	7	8	9	A	B	F	F	F	F	F	F	F	F	F	5	6	7	8	9	A	B	C	D	E	F
53	0	1	2	3	4	5	6	7	8	9	A	B	F	F	F	F	F	F	F	F	F	5	6	7	8	9	A	B	C	D	E	F
54	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Example 5

This example uses plane masking; the most significant bit is masked. Before instruction execution, PMASK = >8888 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory contains the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	0	1	2	3	4	5	6	7	F	F	F	F	F	F	F	F	7	7	7	7	7	7	7	7	F	9	A	B	C	D	E	F
53	0	1	2	3	4	5	6	7	F	F	F	F	F	F	F	F	7	7	7	7	7	7	7	7	F	9	A	B	C	D	E	F
54	0	1	2	3	4	5	6	7	F	F	F	F	F	F	F	F	7	7	7	7	7	7	7	7	F	9	A	B	C	D	E	F

Syntax **GETPC** <*Rd*>

Execution (PC') → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	0	R	Rd			

Description GETPC increments the PC contents by 16 to point past the GETPC instruction, and copies the value into the destination register. Execution continues with the next instruction. This instruction can be used with the EXGPC and JUMP instructions for quick call on jump operations. GETPC can be used to access relocatable data areas whose position relative to the code area is known at assembly time.

Words 1

Machine States 1,4

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples

Code	Before	After
	PC	A1
`GETPC A1`	>0000 1BD0	>0000 1BE0
`GETPC A1`	>0000 1C10	>0000 1C20

Syntax **GETST** *<Rd>*

Execution (ST) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	0	0	R	Rd			

Description GETST copies the contents of the status register into the destination register.

31	30	29	28	27–26	25	24–22	21	20–12	11	10–6	5	4–0
N	C	Z	V	Res	PBX	Res	IE	Reserved	FE1	FS1	FE0	FS0

Status Register

Words 1

Machine States 1,4

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before	After
	PC	A1
`GETST A1`	>2020 0010	>2020 0010
`GETST A1`	>0000 0010	>0000 0010

Syntax INC <*Rd*>

Execution (Rd) + 1 → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	1	R	Rd			

Description INC adds 1 to the contents of the destination register and stores the result in the destination register. This instruction is an alternate mnemonic for `ADDK 1,Rd`.

Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the ADDC instruction.

Words 1

Machine States 1,4

Status Bits
N *1* if the result is negative, *0* otherwise.
C *1* if there is a carry, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before	After	
	A1	A1	NCZV
`INC A1`	>0000 0000	>0000 0001	0000
`INC A1`	>0000 000F	>0000 0010	0000
`INC A1`	>FFFF FFFF	>0000 0000	0110
`INC A1`	>FFFF FFFE	>FFFF FFFF	1000
`INC A1`	>7FFF FFFF	>8000 0000	1001

Syntax **JAcc** *<Address>*

Execution If condition *true*, then Address → PC
If condition *false*, then go to next instruction

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Code				1	0	0	0	0	0	0	0
Address (LSW)															
Address (MSW)															

Operands

cc is a condition mnemonic such as UC, LO, etc. (see condition codes table).

Address is a 32-bit absolute address.

Fields

Code is a 4-bit digit (see condition codes table below).

Description If the specified condition is **true**, jump to the address contained in the two words of extension and continue execution from that point. If the specified condition is **false**, continue execution at the next sequential instruction. Note that the lower four bits of the program counter are set to 0 (word aligned). These instructions are usually used in conjunction with the CMP and CMPI instructions. The JAV and JANV instructions can also be used to detect window violations or CPW status.

Condition Codes

Mnemonic†	Code	Condition	Status Bits
JAUC	0000	Unconditional	No conditions
Unsigned Compare			
JALO (JAC)	1000	Lower than	C
JALS	0010	Lower or same	C + Z
JAHI	0011	Higher than	$\overline{C} \cdot \overline{Z}$
JAHS (JANC)	1001	Higher or same	$\overline{C}$
JAEQ (JAZ)	1010	Equal	Z
JANE (JANZ)	1011	Not equal	$\overline{Z}$
Signed Compare			
JALT	0100	Less than	$(N \cdot \overline{V}) + (\overline{N} \cdot V)$
JALE	0110	Less than or equal	$(N \cdot \overline{V}) + (\overline{N} \cdot V) + Z$
JAGT	0111	Greater than	$(N \cdot V \cdot \overline{Z}) + (\overline{N} \cdot \overline{V} \cdot \overline{Z})$
JAGE	0101	Greater than or equal	$(N \cdot V) + (\overline{N} \cdot \overline{V})$
JAEQ (JAZ)	1010	Equal	Z
JANE (JANZ)	1011	Not equal	$\overline{Z}$
Compare to Zero			
JAZ	1010	Zero	Z
JANZ	1011	Nonzero	$\overline{Z}$
JAP	0001	Positive	$\overline{N} \cdot \overline{Z}$
JAN	1110	Negative	N
JANN	1111	Nonnegative	$\overline{N}$

Condition Codes (continued)

Mnemonic†	Code	Condition	Status Bits
General Arithmetic			
JAZ	1010	Zero	Z
JANZ	1011	Nonzero	$\overline{Z}$
JAC	1000	Carry	C
JANC	1001	No carry	$\overline{C}$
JAB (JAC)	1000	Borrow	C
JANB (JANC)	1001	No borrow	$\overline{C}$
JAV‡	1100	Overflow	V
JANV‡	1101	No overflow	$\overline{V}$

† Jump instructions in parentheses indicate equivalent instructions
‡ Also window clipping
\+ Logical OR
· Logical AND
‾ Logical NOT

Words 3

Machine States
3,6 (Jump)
4,7 (No jump)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code		Flags for Branch NCZV	NCZV	NCZV	Code		Flags for Branch NCZV	NCZV	NCZV
JAUC	HERE	xxxx			JAV	HERE	xxx1		
JAP	HERE	0x0x			JANZ	HERE	xx0x		
JALS	HERE	xx1x	x1xx		JANN	HERE	0xxx		
JAHI	HERE	x00x			JANV	HERE	xxx0		
JALT	HERE	0xx1	1xx0		JAN	HERE	1xxx		
JAGE	HERE	0xx0	1xx1		JAB	HERE	x1xx		
JALE	HERE	0xx1	1xx0	xx1x	JANB	HERE	x0xx		
JAGT	HERE	0x00	1x01		JALO	HERE	x1xx		
JAC	HERE	x1xx			JAHS	HERE	x00x	xx1x	
JANC	HERE	x0xx			JANE	HERE	xx0x		
JAZ	HERE	xx1x			JAEQ	HERE	xx1x		

Note:

The TMS34010 assembler will take the jump when any one or more of the *Flags for Branch* listed above are set as indicated.

JRcc Jump Relative Conditional - *±127 Words* JRcc

Syntax **JRcc** *<Address>*

Execution If condition *True* then Displacement + (PC') → PC
If condition *False* then go to next instruction

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	code				Displacement							

Operands

cc is a condition mnemonic such as UC, LO, etc. (see condition codes table).

Address is a 32-bit relative address, ±127 words (excluding 0).

Fields

Code is a 4-bit digit (see condition codes table below).

Description

If the condition specified is **true**, then jump to the location at the address specified by the sum of the next instruction address (PC') and the signed word displacement. If the specified condition is **false**, then continue execution at the next sequential instruction.

The displacement is the number of words relative to the PC and is computed by the assembler as (Address - PC')/16. The assembler will use this opcode if the address in the range -127 to 127 words (except for 0). If the displacement is outside the legal range, the assembler will automatically use the longer JRcc instruction. If the displacement is 0, the assembler will automatically substitute a NOP opcode instead. The assembler will **not** accept an address which is externally defined or an address which is relative to a different section than the PC. Note that the four LSBs of the program counter are always 0 (word aligned).

These instructions are usually used in conjunction with the CMP and CMPI instructions. The JRV and JRNV instructions can also be used to detect window violations or CPW status.

Condition Codes

Mnemonic†	Code	Condition	Status Bits
JRUC	0000	Unconditional	No conditions
Unsigned Compare			
JRLO (JRC)	1000	Lower than	C
JRLS	0010	Lower or same	C + Z
JRHI	0011	Higher than	$\overline{C} \cdot \overline{Z}$
JRHS (JRNC)	1001	Higher or same	$\overline{C}$
JREQ (JRZ)	1010	Equal	Z
JRNE (JRNZ)	1011	Not equal	$\overline{Z}$
Signed Compare			
JRLT	0100	Less than	$(N \cdot \overline{V}) + (\overline{N} \cdot V)$
JRLE	0110	Less than or equal	$(N \cdot \overline{V}) + (\overline{N} \cdot V) + Z$
JRGT	0111	Greater than	$(N \cdot V \cdot \overline{Z}) + (\overline{N} \cdot \overline{V} \cdot \overline{Z})$
JRGE	0101	Greater than or equal	$(N \cdot V) + (\overline{N} \cdot \overline{V})$
JREQ (JRZ)	1010	Equal	Z
JRNE (JRNZ)	1011	Not equal	$\overline{Z}$

Condition Codes (continued)

Mnemonic†	Code	Condition	Status Bits
Compare to Zero			
JRZ	1010	Zero	Z
JRNZ	1011	Nonzero	$\overline{Z}$
JRP	0001	Positive	$\overline{N} \cdot \overline{Z}$
JRN	1110	Negative	N
JRNN	1111	Nonnegative	$\overline{N}$
General Arithmetic			
JRZ	1010	Zero	Z
JRNZ	1011	Nonzero	$\overline{Z}$
JRC	1000	Carry	C
JRNC	1001	No carry	$\overline{C}$
JRB (JRC)	1000	Borrow	C
JRNB (JRNC)	1001	No borrow	$\overline{C}$
JRV‡	1100	Overflow	V
JRNV‡	1101	No overflow	$\overline{V}$

† Jump instructions in parentheses indicate equivalent instructions
‡ Also window
\+ Logical OR
· Logical AND
‾ Logical NOT

Words 1

Machine States 2,5 (Jump)
1,4 (No jump)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Flags for Branch NCZV	NCZV	NCZV	Code	Flags for Branch NCZV	NCZV	NCZV
JRUC HERE	xxxx			JRC HERE	x1xx		
JRP HERE	0x0x			JRNC HERE	x0xx		
JRLS HERE	xx1x	x1xx		JRZ HERE	xx1x		
JRHI HERE	x00x			JRNZ HERE	xx0x		
JRLT HERE	0xx1	1xx0		JRV HERE	xxx1		
JRGE HERE	0xx0	1xx1		JRNV HERE	xxx0		
JRLE HERE	0xx1	1xx0	xx1x	JRN HERE	1xxx		
JRGT HERE	0x00	1x01		JRNN HERE	0xxx		

Note:

The TMS34010 assembler will take the jump when any one or more of the *Flags for Branch* listed above are set as indicated.

Syntax **JRcc** *<Address>*

Execution If condition *True* then Address → PC
If condition *False* then go to next instruction

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	code				0	0	0	0	0	0	0	0
Displacement															

Operands **cc** is a condition mnemonic such as UC, LO, etc. (see condition codes table).

Address is a 32-bit relative address, ±32K words (excluding 0).

Fields **Code** is a 4-bit digit (see condition codes table below).

Description If the specified condition is true, then jump to the location at the address specified by the sum of the next instruction address (PC') and the signed word displacement. If the specified condition is false, then continue execution at the next sequential instruction.

The displacement is the number of words relative to the PC and is computed by the assembler as (Address - PC')/16. The assembler will use this opcode if the displacement is in the range -32,768 to 32,767 words (except for 0). If the displacement is 0, the assembler will automatically substitute a NOP opcode instead. If the address is out of range, the assembler will use the JAcc instruction instead. The assembler will not accept an address which cannot be resolved at assembly time, that is, an address which is externally defined or which is relative to a different section than the current PC. Note that the four LSBs of the program counter are always 0 (word aligned).

These instructions are usually used in conjunction with the CMP and CMPI instructions. The JRV and JRNV instructions can also be used to detect window violations or CPW status.

Condition Codes

Mnemonic†	Code	Condition	Status Bits
JRUC	0000	Unconditional	No conditions
Unsigned Compare			
JRLO (JRC)	1000	Lower than	C
JRLS	0010	Lower or same	C + Z
JRHI	0011	Higher than	$\overline{C} \cdot \overline{Z}$
JRHS (JRNC)	1001	Higher or same	$\overline{C}$
JREQ (JRZ)	1010	Equal	Z
JRNE (JRNZ)	1011	Not equal	$\overline{Z}$
Signed Compare			
JRLT	0100	Less than	$(N \cdot \overline{V}) + (\overline{N} \cdot V)$
JRLE	0110	Less than or equal	$(N \cdot \overline{V}) + (\overline{N} \cdot V) + Z$
JRGT	0111	Greater than	$(N \cdot V \cdot \overline{Z}) + (\overline{N} \cdot \overline{V} \cdot \overline{Z})$
JRGE	0101	Greater than or equal	$(N \cdot V) + (\overline{N} \cdot \overline{V})$
JREQ (JRZ)	1010	Equal	Z
JRNE (JRNZ)	1011	Not equal	$\overline{Z}$

Condition Codes (continued)

Mnemonic†	Code	Condition	Status Bits
Compare to Zero			
JRZ	1010	Zero	Z
JRNZ	1011	Nonzero	$\overline{Z}$
JRP	0001	Positive	$\overline{N} \cdot \overline{Z}$
JRN	1110	Negative	N
JRNN	1111	Nonnegative	$\overline{N}$
General Arithmetic			
JRZ	1010	Zero	Z
JRNZ	1011	Nonzero	$\overline{Z}$
JRC	1000	Carry	C
JRNC	1001	No carry	$\overline{C}$
JRB (JRC)	1000	Borrow	C
JRNB (JRNC)	1001	No borrow	$\overline{C}$
JRV‡	1100	Overflow	V
JRNV‡	1101	No overflow	$\overline{V}$

† Jump instructions in parentheses indicate equivalent instructions
‡ Also window clipping
\+ Logical OR
· Logical AND
‾ Logical NOT

Words 2

Machine States
3,6 (Jump)
2,5 (No jump)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code		Flags for Branch NCZV	NCZV	NCZV	Code		Flags for Branch NCZV	NCZV	NCZV
JRUC	HERE	xxxx			JRZ	HERE	xx1x		
JRP	HERE	0x0x			JRNZ	HERE	xx0x		
JRLS	HERE	xx1x	x1xx		JRV	HERE	xxx1		
JRHI	HERE	x00x			JRNV	HERE	xxx0		
JRLT	HERE	0xx1	1xx0		JRN	HERE	1xxx		
JRGE	HERE	0xx0	1xx1		JRNN	HERE	0xxx		
JRLE	HERE	0xx1	1xx0	xx1x	JRB	HERE	x1xx		
JRGT	HERE	0x00	1x01		JRNB	HERE	x0xx		
JRC	HERE	x1xx			JRLO	HERE	x1xx		
JRNC	HERE	x0xx			JRHS	HERE	x00x	xx1x	

> Note:
>
> The TMS34010 assembler will take the jump when any one or more of the *Flags for Branch* listed above are set as indicated.

Syntax **JUMP** <*Rs*>

Execution (Rs) → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	1	R	Rs			

Operands **Rs** contains the new PC value.

Description JUMP jumps to the address contained in the source register. The TMS34010 sets the four LSBs of the program counter to 0 (word aligned). This instruction can be used in conjunction with the GETPC and/or EXGPC instructions.

Words 1

Machine States 2,5

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples

Code	Before A1	Before PC	After PC
`JUMP A1`	>0000 1EE0	>0055 5550	>0000 1EE0
`JUMP A1`	>0000 1EE5	>0055 5550	>0000 1EE0
`JUMP A1`	>FFFF FFFF	>0055 5550	>FFFF FFF0

Syntax LINE {0,1}

Execution The two execution algorithms for the LINE instruction are explained below. These algorithms are similar, varying only in their treatment of $d=0$.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	1	1	1	Z	0	0	1	1	0	1	0

Operands **Z** is the algorithm select bit:
Z=0 selects algorithm 0.
Z=1 selects algorithm 1.

Description LINE performs the inner loop of Bresenham's line-drawing algorithm. This type of line draw plots a series of points (x_i,y_i) either diagonally or laterally with respect to the previous point. Movement from pixel to pixel always proceeds in a dominant direction. The algorithm may or may not also increment in the direction with the smaller dimension (this produces a diagonal movement). Two XY-format registers supply the XY increment values for the two possible movements. The LINE instruction maintains a decision variable, *d*, that acts as an error term, controlling movement in either the dominant or diagonal direction. The algorithm operates in one of two modes, depending on how the condition $d=0$ is treated. During LINE execution, some portion of a line $[(x_0,y_0)(x_1,y_1)]$ will be drawn. The line is drawn so that the axis with the largest extent has dimension *a* and the axis with the least extent has dimension *b*. Thus, *a* is the larger (in absolute terms) of $y_1 - y_0$ or $x_1 - x_0$ and *b* is the smaller of the two. This means that $a \geq b \geq 0$.

The following values must be supplied to draw a line from (x_0,y_0) to (x_1,y_1):

1) Set the XY pointer (x_i,y_i) in the DADDR register to the initial value of (x_0,y_0).

2) Use the line endpoints to determine the major and minor dimensions (*a* and *b*, respectively) for the line draw; then set the DYDX register to this value (*b*:*a*).

3) Place the signed XY increment for a movement in the diagonal (or minor) direction ($d \geq 0$ for Z=0, $d > 0$ for Z=1) in the INC1 register.

4) Place the signed XY increment for a movement in the dominant (or major) direction ($d < 0$ for Z=0, $d \leq 0$ for Z=1) in the INC2 register.

5) Set the initial value of the decision variable in register B0 to $2b - a$.

6) Set the initial count value in the COUNT register to $a + 1$.

7) Set the LINE color in the COLOR1 register.

8) Set the PATTRN register to all 1s.

The LINE instruction may use one of two algorithms, depending on the value of Z.

Algorithm 0 (Z=0):

```
While COUNT > 0
    Draw the next pixel
    If d ≥ 0
        d = d + 2b - 2a
        POINTER = POINTER + INC1
    Else d = d + 2b;
        POINTER = POINTER + INC2
```

Algorithm 1 (Z=1):

```
While COUNT > 0
    Draw the next pixel
    If d > 0
        d = d + 2b - 2a
        POINTER = POINTER + INC1
    Else d = d + 2b;
        POINTER = POINTER + INC2
```

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B0†	SADDR	Integer	Decision variable, *d*
B2†	DADDR	XY	Starting point (y_i:x_i), usually (y_0:x_0)
B4	OFFSET	Linear	Screen origin (0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner
B7	DYDX	XY	*b*:*a* minor :major line dimensions
B9	COLOR1	Pixel	Pixel color to be replicated
B10†	COUNT	Integer	Loop count
B11	INC1	XY	Minor axis (diagonal) increment, INC1
B12	INC2	XY	Major axis (dominant) increment, INC2
B13†	PATTRN	Pattern	Future pattern register, must be set to all 1s
B15	TEMP	–	Temporary register

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP – Pixel processing operations W – Window clipping operation T – Transparency operation
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† These registers are changed by instruction execution

Pixel Processing

The PP field in the CONTROL I/O register specifies the operation to be applied to the pixel as it is written. There are 22 operations; the default case at reset is the pixel processing *replace* (S → D) operation. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Window Checking

Window clipping or pick is selected by setting the W bits in the CONTROL I/O register to the appropriate value. The WSTART and WEND registers define the window in XY-coordinate space.

Options include:

0 *No window clipping.* LINE draws the entire line. Neither the WVP or V bit are affected. WSTART and WEND are ignored.

1 *Window hit.* The instruction calculates points but no pixels are actually drawn. As soon as the pixel to be drawn lies inside the window, the WVP bit is set, the V bit is cleared, and the instruction is aborted. If the line lies entirely outside the window, then the WVP bit is not affected, the V bit in the status is set, and the instruction completes execution.

2 *Clip and set WVP.* LINE draws pixels until the pixel to be drawn lies outside the window. At this point, the WVP bit is set, the V bit is set, and the instruction is aborted. If the entire line lies within the window, then the WVP bit **is not affected**, the V bit is cleared and the instruction completes execution. The initial value of WVP does not affect instruction execution.

3 *Clip.* LINE calculates all the points, but only draws the points that lie inside the window. The V bit tracks the state of the last pixel. If the pixel was outside the window, V is set to 1; otherwise, it is 0. The instruction will traverse the entire line.

The default case at reset is no window clipping. For more information, see Section 7.10, Window Checking, on page 7-25.

Transparency

Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it processes the source data. At reset, the default case for transparency is *off.*

Plane Mask

The plane mask is enabled for this instruction.

Interrupts

LINE may be interrupted after every pixel in the line draw except for the last pixel. If the instruction is interrupted, the PC is decremented by 16 to point back to the LINE instruction (the one being executed) before the PC is pushed on the stack. Thus, the LINE instruction will be resumed upon return from the interrupt. In order for the LINE to be resumed correctly, any B-file registers that are modified by the interrupting routine must be restored, and the RETI or RETS instruction must be executed. Note that a LINE instruction that is aborted because of window checking options 1 or 2 does not decrement the PC before pushing it on the stack. In this case, the LINE is not resumed after returning from the interrupt service routine.

Words

1

Machine States See Section 13.6, The LINE Instruction.

Status Bits
- **N** Undefined
- **C** Undefined
- **Z** Undefined
- **V** Set depending upon window operation.

Linedraw Code

The following code segment shows setup and execution of the LINE instruction.

```
            .file    'LineDraw'
            .globl   _draw_line
            .globl   _xyorigin

_draw_line:
            MMTM     SP,B2,B7,B10,B11,B12,B13,B14

            MOVE     A14,B14
            MOVE     *-B14,B2,1        ; Get starting x
            MOVE     *-B14,B11,1       ; Get starting y
            SLL      16,B11
            MOVY     B11,B2            ; B2 = (y0,x0)
            MOVE     *-B14,B10,1       ; Get ending x
            MOVE     *-B14,B11,1       ; Get ending y
            SLL      16,B11
            MOVY     B11,B10           ; B10 = (y1,x1)
            MOVE     B14,A14

            MOVE     @_xyorigin,B11,1
            ADDXY    B11,B2            ; Add viewport offset
            ADDXY    B11,B10           ; Add viewport offset

draw_line:                             ; Draw line from (y0,x0) to (y1,x1)
            CLR      B7                ; B2 = (y0,x0), B10 = (y1,x1)
            SUBXY    B2,B10            ; B10 = (y1-y0,x1-x0) = (b,a)
            JRZ      horiz_line
            JRN      vert_line
            JRNC     bpos
            JRNV     bneg_apos
bneg_aneg:  SUBXY    B10,B7            ; B7 = (|b|,|a|)
            MOVI     -1,B11            ; B11 = (-1,-1)
            JRUC     cmp_b_a
bneg_apos:  SUBXY    B10,B7
            MOVX     B10,B7            ; B7 = (|b|,|a|)
            MOVI     >FFFF0001,B11     ; B11 = (-1,1)
            JRUC     cmp_b_a
bpos:       JRNV     bpos_apos
bpos_aneg:  SUBXY    B10,B7
            MOVY     B10,B7            ; B7 = (|b|,|a|)
            MOVI     >0001FFFF,B11     ; B11 = (1,-1)
            JRUC     cmp_b_a
bpos_apos:  MOVE     B10,B7            ; B7 = (|b|,|a|)
            MOVI     >00010001,B11     ; B11 = (1,1)
```

```
cmp_b_a:    CLR     B12
            MOVI    -1,B13          ; B13 = FFFFFFFF (set pattern to
                                    ; all 1s)
            MOVE    B7,B0
            SRL     16,B0           ; B0 = b
            CLR     B10
            MOVX    B7,B10          ; B10 = a
            CMP     B0,B10
            JRGT    a_ge_b
a_lt_b:     MOVE    B0,B10
            MOVX    B7,B0
            RL      16,B7           ; a and b swapped
            MOVY    B11,B12
            SLL     1,B0
            SUB     B10,B0          ; B0 = 2b - a
            ADDK    1,B10
            MOVE    B11,B11         ; If drawing in +Y direction, use
            JRN     line1           ; LINE 0, otherwise use LINE 1
line0:      LINE    0
            JRUC    done

a_ge_b:     MOVX    B11,B12
            SLL     1,B0
            SUB     B10,B0          ; B0 = 2b - a
            MOVE    B11,B11         ; If drawing in -Y direction, use
            JRNN    line0           ; LINE 1, otherwise use LINE 0
line1:      LINE    1
            JRUC    done

horiz_line: JRN     pixel
            JRNV    do_fill
            SUBXY   B10,B7          ; Make DX positive
            MOVE    B7,B10
            ADDXY   B10,B2          ; Change start to (y1,x1)

vert_line:  JRNC    do_fill
            NEG     B10             ; Make DY positive
            ADDXY   B10,B2          ; Change start to (y1,x1)

do_fill:    MOVE    B10,B7
            ADDI    >10001,B7
            FILL    XY
            JRUC    done

pixel:      DRAV    B12,B2

done:       MMFM    SP,B2,B7,B10,B11,B12,B13,B14
            RETS    2               ; Return to calling routine
```

Example 1

This example draws a line from (3,52) to (19,55). Window checking is off, transparency and the pixel processing replace operation are selected, and plane masking is disabled. Assume the following registers *have been* loaded with these values:

B0	=	>FFFF FFF1	Decision variable $d = 2b - a = -15$
B2	=	>0052 0003	DADDR
B3	=	>0000 0800	DPTCH (CONVDP=13)
B4	=	>0000 0100	OFFSET
B5	=	>0030 0003	WSTART
B6	=	>0055 0025	WEND
B7	=	>0003 0016	*b*:*a*; *b*=3 and *a*=22
B9	=	>4444 4444	COLOR1 (color of the line)
B10	=	>0000 0017	COUNT (*a*+1)
B11	=	>0001 0001	Diagonal increment (+1,+1)
B12	=	>0000 0001	Nondiagonal increment (0,+1)
B13	=	>FFFF FFFF	PATTRN (all 1s)

This line is shown in Figure 12-11, represented by ●s.

Before LINE execution, DADDR contains the first pixel to be drawn. During LINE execution, DADDR is updated so that it always points to the next pixel to be drawn. After this example is completed, DADDR will equal >0055 001A. Register B7 contains the X and Y dimensions of the line. Register B10 indicates the number of pixels that will be drawn; if you want the endpoint to be drawn (in this case, (19,55)), B10 should equal *a*+1.

B11 contains the XY increment for diagonal moves. You can see the line progressing in a diagonal direction when it moves from (6,52) to (7,53); it is incremented by 1 in both the X and the Y dimensions. B12 contains the XY increment for nondiagonal moves. You can see the line progressing in a nondiagonal direction when it moves from (3,52) to (4,52); it is incremented by 1 in the X dimension.

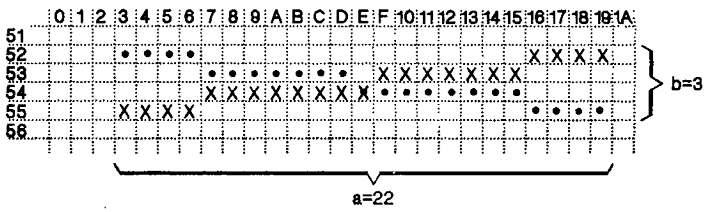


Figure 12-11. LINE Examples

Example 2 This example draws a line from (19,52) to (3,55). Window checking is off, transparency and the pixel processing replace operation are selected, and plane masking is disabled. Assume the following registers have been loaded with these values:

B0	= >FFFF FFF1	Decision variable $d = 2b - a = -15$
B2	= >0052 0019	DADDR
B3	= >0000 0800	DPTCH (CONVDP=13)
B4	= >0000 0100	OFFSET
B5	= >0030 0003	WSTART
B6	= >0055 0025	WEND
B7	= >0003 0016	b:a; b=3 and a=22
B9	= >2222 2222	COLOR1 (color of the line)
B10	= >0000 0017	COUNT (a+1)
B11	= >0001 FFFF	Diagonal increment (+1,-1)
B12	= >0000 FFFF	Nondiagonal increment (0,-1)
B13	= >FFFF FFFF	PATTRN (all 1s)

This line is shown in Figure 12-11, represented by Xs.

Before LINE execution, DADDR contains the first pixel to be drawn. During LINE execution, DADDR is updated so that it always points to the next pixel to be drawn. After this example is completed, DADDR will equal >0055 0002. Register B7 contains the X and Y dimensions of the line. Register B10 indicates the number of pixels that will be drawn; if you want the endpoint to be drawn (in this case, (3,55)), B10 should equal a+1.

B11 contains the XY increment for diagonal moves. You can see the line progressing in a diagonal direction when it moves from (F,53) to (E,54); it is decremented by 1 in the X dimension and incremented by 1 in the Y dimension. B12 contains the XY increment for nondiagonal moves. You can see the line progressing in a nondiagonal direction when it moves from (14,53) to (13,53); it is decremented by 1 in the X dimension.

LMO Leftmost One LMO

Syntax **LMO** *<Rs>,<Rd>*

Execution 31 - (Bit number of leftmost 1 bit in Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	Rs				R	Rd			

Operands **Rs** is the register to be evaluated.

Description LMO locates the leftmost (most significant) 1 in the source register. It then loads the 1's complement of the **bit number** of the leftmost-1 bit into the five LSBs of the destination register. The 27 MSBs of the destination register are loaded with 0s. Bit 31 of Rs is the MSB (leftmost) and bit 0 is the LSB. If there are no 1 bits in the source register, then the destination result is 0 and status bit Z is set.

The source register contents can be normalized by following this instruction by executing the `RL Rs,Rd` instruction, where Rs is the destination register of the LMO instruction and Rd is the source register.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** *1* if the source register contents are 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before	After	
	A0	NCZV	A1
`LMO A0,A1`	>0000 0000	xx1x	>0000 0000
`LMO A0,A1`	>0000 0001	xx0x	>0000 001F
`LMO A0,A1`	>0000 0010	xx0x	>0000 001B
`LMO A0,A1`	>0800 0000	xx0x	>0000 0004
`LMO A0,A1`	>8000 0000	xx0x	>0000 0000

Syntax MMFM *<Rs>*,[*<register list>*]

Execution If Register *n* in <register list> then *Rs+ → R*n*
Repeat for *n* = 0 to 15

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	0	1	R	Rs			
Mask															

Operands **Rs** points to the first location in a block of memory.

Register list is a list of registers to be moved (such as A0,A1,A9).

Fields **Mask** is a binary representation of the register list.

Description MMFM loads the contents of a specified list of *either* A or B file registers (not both) from a block of memory. Rs points to the first location in the memory block. Rs and the registers in the list must be in the same register file.

The MMFM and MMTM instructions can be thought of as "stack" instructions for storing and retrieving multiple registers in memory. MMTM stores the registers in memory, using Rs as a "stack pointer." The stack "shrinks" in the direction of increasing linear address, with Rs containing the bit address of the top of the stack. MMFM reverses the action of the MMTM instruction. Rs is postincremented by 32 when popping off the stack. Each register is removed from the stack LSW first, with higher order registers moved first. (The alignment of Rs affects the instruction timing as indicated in **Machine States**, below.) If a 0 mask is supplied, the SP will be popped from memory and loaded. Note that including Rs in the register list produces unpredictable results.

The bit assignments in the mask are:

If Rs is in file A:

SP	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
15(MSB)															0(LSB)

If Rs is in file B:

SP	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
15(MSB)															0(LSB)

Words 2

Machine States

	Cache Enabled	Cache Disabled
Aligned:	3 + 4n + (2) extended states	9 + 4n
Nonaligned:	3 + 8n + (6) extended states	9 + 8(n + 1)

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples Assume that memory contains the following values before *instruction* execution:

Address	Data	Address	Data
>000100F0	>1111	>00010070	>CCCC
>000100E0	>B1B1	>00010060	>BCBC
>000100D0	>2222	>00010050	>DDDD
>000100C0	>B2B2	>00010040	>BDBD
>000100B0	>3333	>00010030	>EEEE
>000100A0	>B3B3	>00010020	>BEBE
>00010090	>7777	>00010010	>FFFF
>00010080	>B7B7	>00010000	>BFBF

Register B0 = >0001 0000

```
MMFM B0,B1,B2,B3,B7,B12,B13,B14,SP
```
or
```
MMFM B0,>710F
```

Register contents after instruction execution:

B0 = >0010 0100	B12 = >CCCC BCBC
B1 = >1111 B1B1	B13 = >DDDD BDBD
B2 = >2222 B2B2	B14 = >EEEE BEBE
B4 = >3333 B3B3	SP = >FFFF BFBF
B8 = >7777 B7B7	Others unchanged

MMTM Move Multiple Registers to Memory MMTM

Syntax **MMTM** *<Rd>,<register list>*

Execution If Register *n* in <register list> then R*n* → -*Rd
Repeat for *n* = 0 to 15

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	0	0	R	Rd			
Mask															

Operands **Register list** is a list of registers to be moved (such as A0,A1,A9).

Fields **Mask** is a binary representation of the register list.

Description MMTM stores the contents of a specified list of *either* A or B file registers (not both) from a block of memory. Rs points to the first location in the memory block. Rs and the registers in the list must be in the same register file.

The MMFM and MMTM instructions can be thought of as "stack" instructions for storing and retrieving multiple registers in memory. MMTM stores the registers in memory, using Rs as a "stack pointer." The stack "shrinks" in the direction of increasing linear address, with Rs containing the bit address of the top of the stack. MMFM reverses the action of the MMTM instruction. Rs is postincremented by 32 when popping off the stack. Each register is removed from the stack LSW first, with higher order registers moved first. (The alignment of Rs affects the instruction timing as indicated in **Machine States**, below.)

When execution of the MMTM instruction is complete, the contents of the lowest-numbered register in the list will reside at the highest address in the memory block. Rd will have been decremented to point to the contents of the highest-numbered register in the list.

If a register list is not specified, the GSP will store **all** the registers of a register file, starting at the location specified by Rs. Rs indicates the register file that will be affected. For example, `MMTM A3` stores the A-file registers in memory, beginning at the address in register A3. Similarly, `MMTM B0` stores the B-file registers in memory, beginning at the address in register B0. If you use SP as the pointer register in this manner, the GSP will assume you want to store the A-file registers inm memory. If you want to use the stack pointer but intend to store the B-file registers, use B15 instead of SP.

The GSP uses a mask to indicate which registers will be affected. Registers in the list are indicated by a 1 in the appropriate location within the mask. If a 0 mask is supplied, A0 or B0 will be pushed on the stack. The bit assignments in the mask are:

If Rs is in file A:

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	SP

15(MSB) 0(LSB)

If Rs is in file B:

B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	SP

15(MSB) 0(LSB)

Words 2

Machine States

	Cache Enabled	**Cache Disabled**
Aligned:	2 + 4n + (2)	8 + 4n + 2
Nonaligned:	2 + 10n + (8)	10(n + 1)

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Assume that these registers contain the following values before instruction execution:

A1 = >0010 0000	A12 = >CCCC ACAC
A0 = >0000 A0A0	A13 = >DDDD ADAD
A2 = >2220 A2A2	A14 = >EEEE AEAE
A4 = >4444 A4A4	SP = >FFFF AFAF
A8 = >8888 A8A8	

```
MMTM  A1,A0,A2,A4,A8,A12,A13,A14,SP
```
or
```
MMTM A1,>A88F
```

After instruction execution, register A1 = >000F FF00. The other registers are not changed.

Memory will contain the following values after instruction execution:

Address	Data	Address	Data
>000FFF00	>AFAF	>000FFF80	>A8A8
>000FFF10	>FFFF	>000FFF90	>8888
>000FFF20	>AEAE	>000FFFA0	>A4A4
>000FFF30	>EEEE	>000FFFB0	>4444
>000FFF40	>ADAD	>000FFFC0	>A2A2
>000FFF50	>DDDD	>000FFFD0	>2222
>000FFF60	>ACAC	>000FFFE0	>A0A0
>000FFF70	>CCCC	>000FFFF0	>0000

Syntax **MODS** *<Rs>,<Rd>*

Execution (Rd) mod (Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	Rs				R	Rd			

Description MODS performs a 32-bit signed divide of the 32-bit dividend in the destination register by the 32-bit value in the source register, and returns a 32-bit remainder in the destination register. The remainder is the same sign as the dividend. The original contents of the destination register will always be overwritten.

The source and destination registers must be in the same register file.

Words 1

Machine States

40,43 (normal case)
41,44 if result = 80000000
3,6 if Rs = 0

Status Bits

N *1* if the remainder is negative, *0* otherwise.
C Unaffected
Z *1* if the remainder is 0, *0* otherwise.
V *1* if the quotient overflows (cannot be represented by 32 bits), *0* otherwise. The following conditions set the overflow flag:

- The divisor is 0
- The quotient cannot be contained within 32 bits

Examples

Code	Before A0	Before A1	After NCZV	After A0
MODS A0,A1	>0000 0000	>0000 0000	0x01	>0000 0000
MODS A0,A1	>0000 0000	>0000 0007	0x01	>0000 0007
MODS A0,A1	>0000 0000	>FFFF FFF9	0x01	>FFFF FFF9
MODS A0,A1	>0000 0004	>0000 0008	0x10	>0000 0000
MODS A0,A1	>0000 0004	>0000 0007	0x00	>0000 0003
MODS A0,A1	>0000 0004	>0000 0000	0x10	>0000 0000
MODS A0,A1	>0000 0004	>FFFF FFF9	1x00	>FFFF FFFD
MODS A0,A1	>0000 0004	>FFFF FFF8	0x10	>0000 0000
MODS A0,A1	>FFFF FFFC	>0000 0008	0x10	>0000 0000
MODS A0,A1	>FFFF FFFC	>0000 0007	0x00	>0000 0003
MODS A0,A1	>FFFF FFFC	>0000 0000	0x10	>0000 0000
MODS A0,A1	>FFFF FFFC	>FFFF FFF9	1x00	>FFFF FFFD
MODS A0,A1	>FFFF FFFC	>FFFF FFF8	0x10	>0000 0000

Syntax **MODU** *<Rs>*,*<Rd>*

Execution (Rd) mod (Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	Rs				R	Rd			

Description MODU performs a 32-bit unsigned divide of the 32-bit dividend in the destination register by the 32-bit value in the source register, and returns a 32-bit remainder in the destination register. The original contents of the destination register will always be overwritten.

The source and destination registers must be in the same register file.

Words 1

Machine States 35,38
3,6 if Rs = 0

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** *1* if the remainder is 0, *0* otherwise.
- **V** *1* if divisor (Rs) equals 0, *0* otherwise.

Examples

Code	Before A0	Before A1	After NCZV	After A1
`MODU A0,A1`	>0000 0000	>0000 0000	xx01	>0000 0000
`MODU A0,A1`	>0000 0000	>0000 0007	xx01	>0000 0007
`MODU A0,A1`	>0000 0000	>FFFF FFF9	xx01	>FFFF FFF9
`MODU A0,A1`	>0000 0004	>0000 0008	xx10	>0000 0000
`MODU A0,A1`	>0000 0004	>0000 0007	xx00	>0000 0003
`MODU A0,A1`	>0000 0004	>0000 0000	xx10	>0000 0000
`MODU A0,A1`	>0000 0004	>FFFF FFF9	xx00	>0000 0001

Syntax **MOVB** *<Rs>*,**<Rd>*

Execution Rs → *Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	Rs				R	Rd			

Operands

Rs The source byte is the eight LSBs of the register.

***Rd** The destination location is the memory address contained in the specified register.

Description MOVB moves a byte from the source register to the memory address contained in the destination register. The source operand byte is right justified in the source register and it is the eight LSBs of the register which are moved. The memory address is a bit address and the field size for the move is eight bits. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>5000	>0000
>5010	>0000

Code	Before A0	Before A1	After @>5000	After @>5010
`MOVB A0,*A1`	>89AB CDEF	>0000 5000	>00EF	>0000
`MOVB A0,*A1`	>89AB CDEF	>0000 5001	>01DE	>0000
`MOVB A0,*A1`	>89AB CDEF	>0000 5009	>DE00	>0001
`MOVB A0,*A1`	>89AB CDEF	>0000 500C	>F000	>000E

MOVB — Move Byte - *Register to Indirect with Displacement* — MOVB

Syntax **MOVB** *<Rs>,*<Rd(Displacement)>*

Execution Rs → *(Rd + Displacement)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	Rs				R	Rd			
Displacement															

Operands

Rs The source byte is the eight LSBs of the register.

***Rd(Displacement)**
The destination location is the memory address formed by the sum of the specified register contents and the signed 16-bit displacement, contained in the extension word following the opcode.

Description MOVB moves a byte from the source register to the destination memory address. The source operand byte is right justified in the source register; it is the eight LSBs of the register which are moved. The destination memory address is a bit address and is formed by adding the contents of the specified register to the signed 16-bit displacement. This is a field move, and the field size for the move is eight bits. The source and destination registers must be in the same register file.

Words 2

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	**Data**
>10000	>0000
>10010	>0000

Code	Before A0	Before A1	After @>10000	After @>10010
`MOVB A0,*A1(0)`	>89AB CDEF	>0001 0000	>00EF	>0000
`MOVB A0,*A1(1)`	>89AB CDEF	>0001 0000	>01DE	>0000
`MOVB A0,*A1(9)`	>89AB CDEF	>0001 0000	>DE00	>0001
`MOVB A0,*A1(12)`	>89AB CDEF	>0001 0000	>F000	>000E
`MOVB A0,*A1(32767)`	>89AB CDEF	>0000 8001	>00EF	>0000
`MOVB A0,*A1(-32768)`	>89AB CDEF	>0001 8000	>00EF	>0000

Syntax **MOVB** *<Rs>,@<DAddress>*

Execution Rs → @DAddress

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1	1	1	R	Rs			
Destination Address (LSW)															
Destination Address (MSW)															

Operands

Rs The source byte is the eight LSBs of the register.

DAddress
The destination location is the linear memory address contained in the two extension words following the instruction.

Description MOVB moves a byte from the source register to the destination memory address. The source operand byte is right justified in the source register and it is the eight LSBs of the register which are moved. The specified destination memory address is a bit address and the field size for the move is eight bits. The source and destination registers must be in the same register file.

Words 3

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	**Data**
>5000	>0000
>5010	>0000

Code	Before	After	
	A0	**@>5000**	**@>5010**
`MOVB A0,@>5000`	>89AB CDEF	>00EF	>0000
`MOVB A0,@>5001`	>89AB CDEF	>01DE	>0000
`MOVB A0,@>5009`	>89AB CDEF	>DE00	>0001
`MOVB A0,@>500C`	>89AB CDEF	>F000	>000E

Syntax **MOVB** *<Rs>,<Rd>*

Execution *Rs → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	Rs				R	Rd			

Operands ***Rs** The source byte location is the memory address contained in the specified register.

Description MOVB moves a byte from the memory address contained in the source register to the destination register. The source memory address is a bit address and the field size for the move is eight bits. When the byte is moved into the destination register, it is right justified and sign extended to 32 bits. This instruction also performs an implicit compare to 0 of the field data. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N *1* if the sign-extended data moved into register is negative, *0* otherwise.
C Unaffected
Z *1* if the sign-extended data moved into register is 0, *0* otherwise.
V *0*

Examples Assume that memory contains the following values before instruction execution:

Address	**Data**
>5000	>00EF
>5010	>89AB

Code	Before A0	After A1	NCZV
`MOVB *A0,A1`	>0000 5000	>FFFF FFEF	1x00
`MOVB *A0,A1`	>0000 5001	>0000 0077	0x00
`MOVB *A0,A1`	>0000 5008	>0000 0000	0x10
`MOVB *A0,A1`	>0000 500C	>FFFF FFB0	1x00

Syntax **MOVB** **<Rs>,*<Rd>*

Execution *Rs → *Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	Rs				R	Rd			

Operands

***Rs** The source byte location is the memory address contained in the specified register.

***Rd** The destination location is the memory address contained in the specified register.

Description MOVB moves a byte from the source memory address to the destination memory address. Both memory addresses are bit addresses and the field size for the move is eight bits. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>5000	>CDEF
>5010	>89AB
>6000	>0000
>6010	>0000

Code	Before A0	A1	After @>6000	@>6010
`MOVB *A0,*A1`	>0000 5000	>0000 6000	>00EF	>0000
`MOVB *A0,*A1`	>0000 5000	>0000 6001	>01DE	>0000
`MOVB *A0,*A1`	>0000 5000	>0000 6009	>DE00	>0001
`MOVB *A0,*A1`	>0000 5000	>0000 600C	>F000	>000E
`MOVB *A0,*A1`	>0000 5001	>0000 6000	>00F7	>0000
`MOVB *A0,*A1`	>0000 5001	>0000 6001	>01EE	>0000
`MOVB *A0,*A1`	>0000 5001	>0000 6009	>EE00	>0001
`MOVB *A0,*A1`	>0000 5001	>0000 600C	>7000	>000F
`MOVB *A0,*A1`	>0000 5009	>0000 6000	>00E6	>0000
`MOVB *A0,*A1`	>0000 5009	>0000 6001	>01CC	>0000
`MOVB *A0,*A1`	>0000 5009	>0000 6009	>CC00	>0001
`MOVB *A0,*A1`	>0000 5009	>0000 600C	>6000	>000E
`MOVB *A0,*A1`	>0000 500C	>0000 6000	>00BC	>0000
`MOVB *A0,*A1`	>0000 500C	>0000 6001	>0178	>0000
`MOVB *A0,*A1`	>0000 500C	>0000 6009	>7800	>0001
`MOVB *A0,*A1`	>0000 500C	>0000 600C	>C000	>000B

MOVB — Move Byte - *Indirect with Displacement to Register* — MOVB

Syntax **MOVB** **<*Rs(Displacement)*>,<*Rd*>*

Execution *(Rs + Displacement) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	Rs				R	Rd			
Displacement															

Operands ***Rs(Displacement)**

The source byte location is the memory address specified by the sum of the specified register contents and the signed 16-bit displacement, contained in the extension word following the opcode.

Description MOVB moves a byte from the source memory address to the destination register. The source memory address is a bit address and is formed by adding the contents of the specified register to the signed 16-bit displacement. The field size is eight bits. When the byte is moved into the destination register, it is right justified and sign extended to 32 bits. This instruction also performs an implicit compare to 0 of the field data. The source and destination registers must be in the same register file.

Words 2

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N *1* if the sign-extended data moved into register is negative, *0* otherwise.
C Unaffected
Z *1* if the sign-extended data moved into register is 0, *0* otherwise.
V *0*

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>10000	>00EF
>10010	>89AB

Code	Before A0	After A1	NCZV
`MOVB *A0(0),A1`	>0001 0000	>FFFF FFEF	1x00
`MOVB *A0(1),A1`	>0001 0000	>0000 0077	0x00
`MOVB *A0(8),A1`	>0001 0000	>0000 0000	0x10
`MOVB *A0(12),A1`	>0001 0000	>FFFF FFB0	1x00
`MOVB *A0(32767),A1`	>0000 8001	>FFFF FFEF	1x00
`MOVB *A0(-32768),A1`	>0001 8000	>FFFF FFEF	1x00

MOVB — Move Byte - *Indirect with Displacement to Indirect with Displacement* — MOVB

Syntax **MOVB** **<*Rs(Displacement)*>, ***<*Rd(Displacement)*>

Execution *(Rs + Displacement) → *(Rd + Displacement)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	Rs				R	Rd			
Source Displacement															
Destination Displacement															

Operands

***Rs(Displacement)**

The source byte location is the memory address specified by the sum of the specified register contents and the signed 16-bit displacement, contained in the first of two extension words following the opcode.

***Rd(Displacement)**

The destination location is the memory address specified by the sum of the specified register contents and the signed 16-bit displacement, contained in the second of two extension words following the opcode.

Description MOVB moves a byte from the source memory address to the destination memory address. Both the source and destination memory addresses are bit addresses and are formed by adding the contents of the specified register to its respective signed 16-bit displacement. The field size is eight bits. The source and destination registers must be in the same register file.

Words 3

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>10000	>CDEF
>10010	>89AB
>11000	>0000
>11010	>0000

Code	Before A0	Before A1	After @>11000	After @>11010
MOVB *A0(0),*A1(0)	>0001 0000	>0001 1000	>00EF	>0000
MOVB *A0(0),*A1(1)	>0001 0000	>0001 1000	>01DE	>0000
MOVB *A0(0),*A1(9)	>0001 0000	>0001 1000	>DE00	>0001
MOVB *A0(0),*A1(12)	>0001 0000	>0001 1000	>F000	>000E
MOVB *A0(0),*A1(32767)	>0001 0000	>0000 9001	>00EF	>0000
MOVB *A0(0),*A1(-32768)	>0001 0000	>0001 9000	>00EF	>0000
MOVB *A0(12),*A1(0)	>0001 0000	>0001 1000	>00BC	>0000
MOVB *A0(12),*A1(1)	>0001 0000	>0001 1000	>0178	>0000
MOVB *A0(12),*A1(9)	>0001 0000	>0001 1000	>7800	>0001
MOVB *A0(12),*A1(12)	>0001 0000	>0001 1000	>C000	>000B
MOVB *A0(12),*A1(32767)	>0001 0000	>0000 9001	>00BC	>0000
MOVB *A0(12),*A1(-32768)	>0001 0000	>0001 9000	>00BC	>0000
MOVB *A0(32767),*A1(0)	>0000 8001	>0001 1000	>00EF	>0000
MOVB *A0(32767),*A1(1)	>0000 8001	>0001 1000	>01DE	>0000
MOVB *A0(32767),*A1(9)	>0000 8001	>0001 1000	>DE00	>0001
MOVB *A0(32767),*A1(12)	>0000 8001	>0001 1000	>F000	>000E
MOVB *A0(32767),*A1(32767)	>0000 8001	>0000 9001	>00EF	>0000
MOVB *A0(32767),*A1(-32678)	>0000 8001	>0001 9000	>00EF	>0000
MOVB *A0(-32768),*A1(0)	>0001 8000	>0001 1000	>00EF	>0000
MOVB *A0(-32768),*A1(1)	>0001 8000	>0001 1000	>01DE	>0000
MOVB *A0(-32768),*A1(9)	>0001 8000	>0001 1000	>DE00	>0001
MOVB *A0(-32768),*A1(12)	>0001 8000	>0001 1000	>F000	>000E
MOVB *A0(-32768),*A1(32767)	>0001 8000	>0000 9001	>00EF	>0000
MOVB *A0(-32768),*A1(-32678)	>0001 8000	>0001 9000	>00EF	>0000

Syntax **MOVB** *@<SAddress>,<Rd>*

Execution @SAddress → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	1	1	1	R	Rd			
Source Address (LSW)															
Source Address (MSW)															

Operands **SAddress**
The source byte location is the linear memory address contained in the two extension words following the instruction.

Description MOVB moves a byte from the source memory address to the destination register. The specified source memory address is a bit address and the field size for the move is eight bits. When the byte is moved into the destination register, it is right justified and sign extended to 32 bits. This instruction also performs an implicit compare to 0 of the field data. The source and destination registers must be in the same register file.

Words 3

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
- **N** *1* if the sign-extended data moved into register is negative, *0* otherwise.
- **C** Unaffected
- **Z** *1* if the sign-extended data moved into register is 0, *0* otherwise.
- **V** *0*

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>10000	>00EF
>10010	>89AB

Code	After A1	NCZV
`MOVB @>10000,A1`	>FFFF FFEF	1x00
`MOVB @>10001,A1`	>0000 0077	0x00
`MOVB @>10008,A1`	>0000 0000	0x10
`MOVB @>1000C,A1`	>FFFF FFB0	1x00

MOVB Move Byte - *Absolute to Absolute* MOVB

Syntax MOVB *@<SAddress>, @<DAddress>*

Execution @SAddress → @DAddress

Encoding

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="16">Source Address (LSW)</td></tr>
<tr><td colspan="16">Source Address (MSW)</td></tr>
<tr><td colspan="16">Destination Address (LSW)</td></tr>
<tr><td colspan="16">Destination Address (MSW)</td></tr>
</table>

Operands

SAddress
The source byte location is the linear memory address contained in the first set of two extension words following the instruction.

DAddress
The destination location is the linear memory address contained in the second set of two extension words following the instruction.

Description MOVB moves a byte from the source memory address to the destination memory address. Both the source and destination addresses are interpreted as bit addresses and the field size for the move is eight bits.

Words 5

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples

Assume that memory contains the following values before instruction execution:

Address	**Data**
>10000	>CDEF
>10010	>89AB
>11000	>0000
>11010	>0000

Code	**After** @>11000	@>11010
MOVB @>10000,@>11000	>00EF	>0000
MOVB @>10000,@>11001	>01DE	>0000
MOVB @>10000,@>11009	>DE00	>0001
MOVB @>10000,@>1100C	>F000	>000E
MOVB @>10001,@>11000	>00F7	>0000
MOVB @>10001,@>11001	>01EE	>0000
MOVB @>10001,@>11009	>EE00	>0001
MOVB @>10001,@>1100C	>7000	>000F
MOVB @>10009,@>11000	>00E6	>0000
MOVB @>10009,@>11001	>01CC	>0000
MOVB @>10009,@>11009	>CC00	>0001
MOVB @>10009,@>1100C	>6000	>000E
MOVB @>1000C,@>11000	>00BC	>0000
MOVB @>1000C,@>11001	>0178	>0000
MOVB @>1000C,@>11009	>7800	>0001
MOVB @>1000C,@>1100C	>C000	>000B

Syntax MOVE *<Rs>,<Rd>*

Execution (Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	M	Rs				R	Rd			

Fields

M Cross File A/File B boundary
M=0 if registers are in same file
M=1 if registers are in different files

R Register file select
R=0 specifies register file A
R=1 specifies register file B

Description MOVE moves the 32 bits of data from the source register to the destination register. Note that this is not a field move; therefore, the field size has no effect. The source and destination registers can be any of the 31 locations in the on-chip register file. Note that this is the only MOVE instruction that allows the source and destination registers to be in different files. This instruction also performs an implicit compare to 0 of the register data.

Words 1

Machine States 1

Status Bits

N *1* if the 32-bit data moved is negative, *0* otherwise.
C Unaffected
Z *1* if the 32-bit data moved is 0, *0* otherwise.
V *0*

Examples

Code	Before	After		
	A0	**A1**	**B1**	**NCZV**
`MOVE A0,A1`	>0000 FFFF	>0000 FFFF	>xxxx xxxx	0x00
`MOVE A0,A1`	>0000 0000	>0000 0000	>xxxx xxxx	0x10
`MOVE A0,A1`	>FFFF FFFF	>FFFF FFFF	>xxxx xxxx	1x00
`MOVE A0,B1`	>0000 FFFF	>xxxx xxxx	>0000 FFFF	0x00
`MOVE A0,B1`	>0000 0000	>xxxx xxxx	>0000 0000	0x10
`MOVE A0,B1`	>FFFF FFFF	>xxxx xxxx	>FFFF FFFF	1x00

Syntax **MOVE** *<Rs>*,**<Rd>*[,*<F>*]

Execution (field)Rs → (field)*Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	F	Rs				R	Rd			

Operands

Rs The source operand is the right justified field in the specified register. 1–32 bits of the register are moved, depending on the field size selected.

***Rd** *Destination register (indirect)*. The destination location is the memory address contained in the specified register.

F is an optional operand; it defaults to 0.
F=**0** selects FS0.
F=**1** selects FS1.

Description MOVE moves a field from the source register to the memory address contained in the destination register. This memory address is a bit address and the field size for the move is 1–32 bits. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	**Data**
>15500	>0000
>15510	>0000
>15520	>0000

Register A0 = >FFFF FFFF

Code	**Before** A1	FS0/1	**After** @>15500	@>15510	@>15520
`MOVE A0,*A1,0`	>0001 5500	5/x	>001F	>0000	>0000
`MOVE A0,*A1,1`	>0001 5503	x/8	>07F8	>0000	>0000
`MOVE A0,*A1,0`	>0001 5508	13/x	>FF00	>001F	>0000
`MOVE A0,*A1,1`	>0001 550B	x/16	>F800	>07FF	>0000
`MOVE A0,*A1,0`	>0001 550D	19/x	>E000	>FFFF	>0000
`MOVE A0,*A1,1`	>0001 550C	x/24	>F000	>FFFF	>000F
`MOVE A0,*A1,0`	>0001 5512	27/x	>0000	>FFFC	>1FFF
`MOVE A0,*A1,1`	>0001 5510	x/32	>0000	>FFFF	>FFFF

Syntax **MOVE** *<Rs>*,***<Rd>*+[,< *F*>]

Execution (field)Rs → (field)*Rd
Rd + field size → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	F	Rs				R	Rd			

Operands

Rs The source operand is the right justified field in the specified register. 1–32 bits of the register which moved, depending on the field size selected.

***Rd+** *Destination register (indirect with postincrement).* The destination location is the memory address contained in the specified register.

F is an optional operand; it defaults to 0.
F=0 selects FS0.
F=1 selects FS1.

Description MOVE moves a field from the source register to the memory address contained in the destination register. The destination register is postincremented after the move by the field size selected. The memory address in the destination register is a bit address and the field size for the move is 1–32 bits. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	**Data**
>15500	>0000
>15510	>0000
>15520	>0000

Register A0 = >FFFF FFFF

<u>**Code**</u>	<u>**Before**</u> A1	FS0/1	<u>**After**</u> A1	@>15500	@>15510	@>15520
`MOVE A0,*A1+,0`	>0001 5528	5/x	>0001 552D	>0000	>0000	>1F00
`MOVE A0,*A1+,1`	>0001 5525	x/8	>0001 552D	>0000	>0000	>1FE0
`MOVE A0,*A1+,0`	>0001 5520	13/x	>0001 552D	>0000	>0000	>1FFF
`MOVE A0,*A1+,1`	>0001 551D	x/16	>0001 552D	>0000	>E000	>1FFF
`MOVE A0,*A1+,0`	>0001 5516	19/x	>0001 5529	>0000	>FFC0	>01FF
`MOVE A0,*A1+,1`	>0001 5507	x/24	>0001 551F	>FF80	>7FFF	>0000
`MOVE A0,*A1+,0`	>0001 5507	27/x	>0001 551F	>FF80	>FFFF	>0003
`MOVE A0,*A1+,1`	>0001 5500	x/32	>0001 5520	>FFFF	>FFFF	>0000

Syntax MOVE *<Rs>,-*<Rd>*[,< *F*>]

Execution Rd - field size → Rd
(field)Rs → (field)*Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	F	Rs				R	Rd			

Operands

Rs The source operand is the right justified field in the specified register. 1–32 bits of the register are moved, depending on the field size.

-*Rd *Destination register (indirect with predecrement).* The destination location is the memory address contained in the specified register predecremented by the field size selected. This is also the final value for the register.

F is an optional operand; it defaults to 0.
F=0 selects FS0.
F=1 selects FS1.

Description MOVE moves a field from the source register to the memory address contained in the destination register predecremented by the field size. The memory address in the destination register is a bit address and the field size for the move is 1–32 bits. The SETF instruction sets the field size and extension. Rs and Rd must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>15500	>0000
>15510	>0000
>15520	>0000

Register A0 = >FFFF FFFF

Code	Before A1	Before FS0/1	After A1	After @>15500	After @>15510	After @>15520
`MOVE A0,-*A1,0`	>0001 5530	5/x	>0001 552B	>0000	>0000	>F800
`MOVE A0,-*A1,1`	>0001 552D	x/8	>0001 5525	>0000	>0000	>1FE0
`MOVE A0,-*A1,0`	>0001 5528	13/x	>0001 551B	>0000	>F800	>00FF
`MOVE A0,-*A1,1`	>0001 5528	x/16	>0001 5518	>0000	>FF00	>00FF
`MOVE A0,-*A1,0`	>0001 5523	19/x	>0001 5510	>0000	>FFFF	>0007
`MOVE A0,-*A1,1`	>0001 5520	x/24	>0001 5508	>FF00	>FFFF	>0000
`MOVE A0,-*A1,0`	>0001 5524	27/x	>0001 5509	>FE00	>FFFF	>000F
`MOVE A0,-*A1,1`	>0001 5520	x/32	>0001 5500	>FFFF	>FFFF	>0000

Syntax　**MOVE** *Rs,*<Rd(Displacement)>*[,< *F*>]

Execution　(field)Rs → (field)*(Rd + Displacement)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	F	Rs				R	Rd			
Displacement															

Operands

Rs　The source operand is the right justified field in the specified register. 1–32 bits of the register are moved, depending on the field size selected.

***Rd(Displacement)**
Destination register with displacement. The destination location is the memory address specified by the sum of the specified register contents and the signed 16-bit displacement, contained in the extension word following the opcode.

F　is an optional operand; it defaults to 0.
F=0 selects FS0.
F=1 selects FS1.

Description　MOVE moves a field from the source register to the destination memory memory address. The destination memory address is a bit address and is formed by adding the contents of the specified register to the signed 16-bit displacement. The field size for the move is 1–32 bits. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words　2

Machine States　See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N　Unaffected
C　Unaffected
Z　Unaffected
V　Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>15530	>0000
>15540	>0000
>15550	>0000

Register A0 = >FFFF FFFF

Code	Before A1	Before FS0/1	After @>15530	After @>15540	After @>15550
`MOVE A0,*A1(>0000),1`	>0001 5530	x/1	>0001	>0000	>0000
`MOVE A0,*A1(>0001),0`	>0001 552F	5/x	>001F	>0000	>0000
`MOVE A0,*A1(>000F),0`	>0001 552D	8/x	>F000	>000F	>0000
`MOVE A0,*A1(>0020),1`	>0001 551C	x/13	>F000	>01FF	>0000
`MOVE A0,*A1(>00FF),0`	>0001 5435	16/x	>FFF0	>000F	>0000
`MOVE A0,*A1(>0FFF),0`	>0001 4531	19/x	>FFFF	>0007	>0000
`MOVE A0,*A1(>7FFF),1`	>0000 D531	x/22	>FFFF	>003F	>0000
`MOVE A0,*A1(>FFF2),1`	>0001 5540	x/25	>FFFC	>07FF	>0000
`MOVE A0,*A1(>8000),0`	>0001 D530	27/x	>FFFF	>07FF	>0000
`MOVE A0,*A1(>FFF0),0`	>0001 5540	31/x	>FFFF	>7FFF	>0000
`MOVE A0,*A1(>FFEC),1`	>0001 5548	x/31	>FFF0	>FFFF	>0007
`MOVE A0,*A1(>FFEC),0`	>0001 554D	32/x	>FE00	>FFFF	>01FF
`MOVE A0,*A1(>001D),0`	>0001 5520	32/x	>E000	>FFFF	>1FFF
`MOVE A0,*A1(>0020),1`	>0001 5520	x/32	>0000	>FFFF	>FFFF

Syntax **MOVE** *<Rs>*,@*<DAddress>*[,< *F*>]

Execution (field)Rs → (field)@DAddress

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	F	1	1	0	0	R	Rs			
Destination Address (LSW)															
Destination Address (MSW)															

Operands

Rs The source operand is the right justified field in the specified register. 1–32 bits of the register are moved, depending on the field size.

DAddress
Linear destination address. The destination location is the memory address contained in the two extension words following the instruction.

F is an optional operand; it defaults to 0.
F=0 selects FS0.
F=1 selects FS1.

Description MOVE moves a field from the source register to the destination memory address. The specified destination memory address is a bit address and the field size for the move is 1–32 bits. SETF sets the field size and extension.

Words 3

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains these values before instruction execution:

Address	Data
>15500	>0000
>15510	>0000
>15520	>0000

Register A0 = >FFFF FFFF

Code	Before FS0/1	After @>15500	@>15510	@>15520
`MOVE A0,@>15500,0`	5/x	>001F	>0000	>0000
`MOVE A0,@>15503,1`	x/8	>07F8	>0000	>0000
`MOVE A0,@>15508,0`	13/x	>FF00	>001F	>0000
`MOVE A0,@>1550B,1`	x/16	>F800	>07FF	>0000
`MOVE A0,@>1550D,0`	19/x	>E000	>FFFF	>0000
`MOVE A0,@>15510,1`	x/24	>0000	>FFFF	>00FF
`MOVE A0,@>15512,0`	27/x	>0000	>FFFC	>1FFF
`MOVE A0,@>1550C,1`	x/32	>F000	>FFFF	>0FFF

Syntax MOVE *<Rs>,<Rd>[,<F>]

Execution (field)*Rs → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	F	Rs				R	Rd			

Operands

***Rs** The source operand location is the memory address contained in the specified register.

F is an optional operand; it defaults to 0.
F=0 selects the FS0, FE0 parameters for the move.
F=1 selects the FS1, FE1 parameters for the move.

Description MOVE moves a field from the memory address contained in the source register to the destination register. The source memory address is a bit address and the field size for the move is 1–32 bits. When the field is moved into the destination register, it is right justified and sign extended or zero extended to 32 bits according to the value of FE. This instruction also performs an implicit compare to 0 of the field data. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N *1* if the field-extended data moved to register is negative, *0* otherwise.
C Unaffected
Z *1* if the field-extended data moved to register is 0, *0* otherwise.
V *0*

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>15500	>7770
>15510	>7777

Register A0 = >0001 5500

Code	Before		After	
	FS0/1	FE0/1	A1	NCZV
MOVE *A0,A1,1	x/1	x/1	>0000 0000	0x10
MOVE *A0,A1,0	5/x	0/x	>0000 0010	0x00
MOVE *A0,A1,1	x/5	x/1	>FFFF FFF0	1x00
MOVE *A0,A1,0	12/x	1/x	>0000 0770	0x00
MOVE *A0,A1,1	x/12	x/0	>0000 0770	0x00
MOVE *A0,A1,0	18/x	0/x	>0003 7770	0x00
MOVE *A0,A1,1	x/18	x/1	>FFFF 7770	1x00
MOVE *A0,A1,0	27/x	1/x	>FF77 7770	1x00
MOVE *A0,A1,1	x/27	x/0	>0777 7770	0x00
MOVE *A0,A1,0	31/x	0/x	>7777 7770	0x00
MOVE *A0,A1,1	x/31	x/1	>F777 7770	1x00
MOVE *A0,A1,0	32/x	x/x	>7777 7770	0x00

Syntax **MOVE** ******<Rs>*,******<Rd>*[,< *F*>]

Execution (field)*Rs → (field)*Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	F	Rs				R	Rd			

Operands

***Rs** The source operand location is the memory address contained in the specified register.

***Rd** The destination location is the memory address contained in the specified register.

F is an optional operand; it defaults to 0.
F=0 selects the FS0 parameter for the move.
F=1 selects the FS1 parameter for the move.

Description MOVE moves a field from the source memory address to the destination memory address. Both memory addresses are bit addresses and the field size for the move is 1-32 bits. The field size is determined by the value of FS for the specified F bit. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data	Address	Data
>15500	>FFFF	>15530	>0000
>15510	>FFFF	>15540	>0000
>15520	>FFFF	>15550	>0000

Code	Before A0	Before A1	After FS0/1	After @>15530	After @>15540	After @>15550
MOVE *A0,*A1,1	>0001 5500	>0001 5530	x/1	>0001	>0000	>0000
MOVE *A0,*A1,0	>0001 5500	>0001 5534	5/x	>01F0	>0000	>0000
MOVE *A0,*A1,1	>0001 5500	>0001 553A	x/10	>FC00	>000 F	>0000
MOVE *A0,*A1,0	>0001 5500	>0001 553F	19/x	>8000	>FFFF	>0003
MOVE *A0,*A1,1	>0001 5504	>0001 5530	x/7	>007F	>0000	>0000
MOVE *A0,*A1,0	>0001 550A	>0001 5530	13/x	>1FFF	>0000	>0000
MOVE *A0,*A1,1	>0001 550D	>0001 5534	x/8	>0FF0	>0000	>0000
MOVE *A0,*A1,0	>0001 550D	>0001 5530	28/x	>FFFF	>0FFF	>0000
MOVE *A0,*A1,1	>0001 5505	>0001 5535	x/23	>FFE0	>0FF F	>0000
MOVE *A0,*A1,0	>0001 5508	>0001 5536	31/x	>FFC0	>FFFF	>001F
MOVE *A0,*A1,1	>0001 5508	>0001 5531	x/31	>FFFE	>FFF F	>0000
MOVE *A0,*A1,0	>0001 550A	>0001 5530	32/x	>FFFF	>FFFF	>0000
MOVE *A0,*A1,0	>0001 5500	>0001 553A	x/32	>FC00	>FFF F	>03FF

Syntax **MOVE** **<*Rs*>+,<*Rd*>[,< *F*>]

Execution (field)*Rs → Rd
(Rs) + field size → Rs

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	F	Rs				R	Rd			

Operands

***Rs+** *Source register (indirect with postincrement).* The source operand location is the memory address contained in the specified register. The register is incremented after the move by the field size selected.

Rd The destination location is the specified register.

F is an optional operand; it defaults to 0.
F=0 selects the FS0, FE0 parameters for the move.
F=1 selects the FS1, FE1 parameters for the move.

Description MOVE moves a field from the memory address contained in the source register to the destination register. The source register is incremented after the MOVE by the field size selected. The source memory address is a bit address and the field size for the move is 1-32 bits. When the field is moved into the destination register, it is right justified and sign extended or zero extended to 32 bits according to the value of FE for the particular F bit selected. This instruction also performs an implicit compare to 0 of the field data. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N *1* if the field-extended data moved to register is negative, *0* otherwise.
C Unaffected
Z *1* if the field-extended data moved to register is 0, *0* otherwise.
V *0*

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>15500	>7770
>15510	>7777

Register A0 = >0001 5500

Code	Before FS0/1	Before FE0/1	After A0	After A1	After NCZV
`MOVE *A0+,A1,1`	x/1	x/0	>0001 5501	>0000 0000	0x10
`MOVE *A0+,A1,1`	x/5	x/0	>0001 5505	>0000 0010	0x00
`MOVE *A0+,A1,0`	5/x	1/x	>0001 5505	>FFFF FFF0	1x00
`MOVE *A0+,A1,0`	12/x	0/x	>0001 550C	>0000 0770	0x00
`MOVE *A0+,A1,1`	x/12	x/1	>0001 550C	>0000 0770	0x00
`MOVE *A0+,A1,0`	18/x	1/x	>0001 5512	>FFFF 7770	1x00
`MOVE *A0+,A1,1`	x/18	x/0	>0001 5512	>0003 7770	0x00
`MOVE *A0+,A1,0`	27/x	0/x	>0001 551B	>0777 7770	0x00
`MOVE *A0+,A1,1`	x/27	x/1	>0001 551B	>FF77 7770	1x00
`MOVE *A0+,A1,0`	31/x	1/x	>0001 551F	>F777 7770	1x00
`MOVE *A0+,A1,1`	x/31	x/0	>0001 551F	>7777 7770	0x00
`MOVE *A0+,A1,0`	32/x	x/x	>0001 5520	>7777 7770	0x00

MOVE — Move Field - *Indirect (Postincrement) to Indirect (Postincrement)* — MOVE

Syntax **MOVE** *****<*Rs*>+, *****<*Rd*>+[,<*F*>]

Execution
(field)*Rs → (field)*Rd
(Rs) + field size → Rs
(Rd) + field size → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	F	Rs				R	Rd			

Operands

***Rs+** *Source Register (indirect with postincrement).* The source operand location is the memory address contained in the specified register. The register is incremented after the move by the field size selected.

***Rd+** *Destination register (indirect with postincrement).* The destination location is the memory address contained in the specified register. The register is postincremented after the move by the field size selected. If Rs and Rd specify the same register, then the destination location is the original contents of the register incremented by twice the FS.

F is an optional operand; it defaults to 0.
F=0 selects the FS0 parameter for the move.
F=1 selects the FS1 parameter for the move.

Description MOVE moves a field from the source memory address to the destination memory address. Both registers are incremented after the move by the field size selected. Both memory addresses are bit addresses and the field size for the move is 1-32 bits. The field size is determined by the value of FS for the F bit specified. The SETF instruction sets the field size and extension. If Rs and Rd specify the same register, the data read from the location pointed to by the original contents of Rs will be written to the location pointed to by the incremented value of Rs (Rd). The source and destination registers must be in the same register file.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data	Address	Data
>15500	>FFFF	>15530	>0000
>15510	>FFFF	>15540	>0000
>15520	>FFFF	>15550	>0000

```
MOVE   *A0+,*A1+,F
```

Before				After				
F	**A0**	**A1**	**FS0/1**	**A0**	**A1**	**@>15530**	**@>15540**	**@>15550**
1	>0001 5500	>0001 553D	x/1	>0001 5501	>0001 553E	>2000	>0000	>0000
0	>0001 5505	>0001 5538	5/x	>0001 550A	>0001 553D	>1F00	>0000	>0000
1	>0001 550A	>0001 553F	x/10	>0001 5514	>0001 5549	>8000	>01FF	>0000
0	>0001 550D	>0001 5530	19/x	>0001 5520	>0001 5543	>FFFF	>0007	>0000
1	>0001 5510	>0001 5532	x/7	>0001 5517	>0001 5539	>01FC	>0000	>0000
0	>0001 5511	>0001 553A	13/x	>0001 551E	>0001 5547	>FC00	>007F	>0000
1	>0001 5513	>0001 553F	x/8	>0001 551B	>0001 5547	>8000	>007F	>0000
0	>0001 5510	>0001 553A	28/x	>0001 552C	>0001 5556	>FC00	>FFFF	>003F
1	>0001 5518	>0001 5534	x/23	>0001 552F	>0001 554B	>FFF0	>07FF	>0000
0	>0001 5510	>0001 5530	31/x	>0001 552F	>0001 554F	>FFFF	>7FFF	>0000
1	>0001 5511	>0001 553D	x/31	>0001 5530	>0001 555C	>E000	>FFFF	>0FFF
0	>0001 5510	>0001 553F	32/x	>0001 5530	>0001 555F	>8000	>FFFF	>7FFF
1	>0001 5500	>0001 5530	x/32	>0001 5520	>0001 5550	>FFFF	>FFFF	>0000

Move Field - *Indirect (Predecrement) to Register*

MOVE

Syntax MOVE -*<*Rs*>,<*Rd*>[,< *F*>]

Execution (Rs) - field size → Rs
(field)*Rs → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	F	Rs				R	Rd			

Operands

-*Rs *Source Register (indirect with predecrement)*. The source operand location is the memory address contained in the specified register decremented by the field size selected. This is also the final value for the register.

F is an optional operand; it defaults to 0.
F=0 selects the FS0, FE0 parameters for the move.
F=1 selects the FS1, FE1 parameters for the move.

Description MOVE moves a field from the memory address contained in the source register to the destination register. The source register is predecremented before the move by the field size selected. The source memory address is a bit address and the field size for the move is 1–32 bits. The field size is determined by the value of FS for the F bit specified. The SETF instruction sets the field size and extension. When the field is moved into the destination register, it is right justified and sign extended or zero extended to 32 bits according to the value of FE for the particular F bit selected. This instruction also performs an implicit compare to 0 of the field data.

The source and destination registers must be in the same register file. If Rs and Rd are the same register, the pointer information is overwritten by the data fetched.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

N *1* if the field-extended data moved to register is negative, *0* otherwise.
C Unaffected
Z *1* if the field-extended data moved to register is 0, *0* otherwise.
V *0*

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>15500	>7770
>15510	>7777

Register A0 = >0001 5520

Code	Before		After		
	FS0/1	FE0/1	A0	A1	NCZV
`MOVE -*A0,A1,1`	x/1	x/0	>0001 551F	>0000 0000	0x10
`MOVE -*A0,A1,0`	5/x	1/x	>0001 551B	>0000 000E	0x00
`MOVE -*A0,A1,1`	x/5	x/0	>0001 551B	>0000 000E	0x00
`MOVE -*A0,A1,0`	12/x	0/x	>0001 5514	>0000 0777	0x00
`MOVE -*A0,A1,1`	x/12	x/1	>0001 5514	>0000 0777	0x00
`MOVE -*A0,A1,0`	18/x	1/x	>0001 550E	>0001 DDDD	0x00
`MOVE -*A0,A1,1`	x/18	x/0	>0001 550E	>0001 DDDD	0x00
`MOVE -*A0,A1,0`	27/x	0/x	>0001 5505	>03BBBBBB	0x00
`MOVE -*A0,A1,1`	x/27	x/1	>0001 5505	>03BBBBBB	0x00
`MOVE -*A0,A1,0`	31/x	1/x	>0001 5501	>3BBBBBB8	0x00
`MOVE -*A0,A1,1`	x/31	x/0	>0001 5501	>3BBBBBB8	0x00
`MOVE -*A0,A1,0`	32/x	x/x	>0001 5500	>7777 7770	0x00

Syntax MOVE *-*<Rs>,-*<Rd>*[,< *F*>]

Execution
(Rs) - field size → Rs
(Rd) - field size → Rd
(field)*Rs → (field)*Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	F	Rs				R	Rd			

Operands

-*Rs *Source Register (indirect with predecrement).* The source operand location is the memory address contained in the specified register decremented by the field size selected. This is also the final value for the register.

-*Rd *Destination register (indirect with predecrement).* The destination location is the memory address contained in the specified register decremented by the field size selected. This is also the final value for the register. If Rs and Rd specify the same register, then the destination location is the original contents decremented by twice the FS.

F is an optional operand; it defaults to 0.
F=0 selects the FS0 parameter for the move.
F=1 selects the FS1 parameter for the move.

Description MOVE moves a field from the source memory address to the destination memory address. Both registers are decremented before the move by the field size selected. Both memory addresses are bit addresses and the field size for the move is 1-32 bits. The field size is determined by the value of FS for the F bit specified. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file. If Rs and Rd are the same register, then the final contents of the register are its original contents decremented by twice the field size.

Words 1

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data	Address	Data
>15500	>FFFF	>15530	>0000
>15510	>FFFF	>15540	>0000
>15520	>FFFF	>15550	>0000

```
MOVE   -*A0,-*A1,F
```

Before				After				
F	**A0**	**A1**	**FS0/1**	**A0**	**A1**	**@>15530**	**@>15540**	**@>15550**
1	>0001 5501	>0001 5531	x/1	>0001 5500	>0001 5530	>0001	>0000	>0000
0	>0001 5505	>0001 5539	5/x	>0001 5500	>0001 5534	>01F0	>0000	>0000
1	>0001 550A	>0001 5544	x/10	>0001 5500	>0001 553A	>FC00	>000F	>0000
0	>0001 5513	>0001 5552	19/x	>0001 5500	>0001 553F	>8000	>FFFF	>0003
1	>0001 550B	>0001 5537	x/7	>0001 5504	>0001 5530	>007F	>0000	>0000
0	>0001 5517	>0001 553D	13/x	>0001 550A	>0001 5530	>1FFF	>0000	>0000
1	>0001 5515	>0001 553C	x/8	>0001 550D	>0001 5534	>0FF0	>0000	>0000
0	>0001 5529	>0001 554C	28/x	>0001 550D	>0001 5530	>FFFF	>0FFF	>0000
1	>0001 551C	>0001 554C	x/23	>0001 5505	>0001 5535	>FFE0	>0FFF	>0000
0	>0001 5527	>0001 5555	31/x	>0001 5508	>0001 5536	>FFC0	>FFFF	>001F
1	>0001 5527	>0001 5550	x/31	>0001 5508	>0001 5531	>FFFE	>FFFF	>0000
0	>0001 552A	>0001 5550	32/x	>0001 550A	>0001 5530	>FFFF	>FFFF	>0000
1	>0001 5520	>0001 555A	x/32	>0001 5500	>0001 553A	>FC00	>FFFF	>03FF

MOVE — Move Field - *Indirect with Displacement to Register* — MOVE

Syntax **MOVE** *<*Rs(Displacement)*>*,< *Rd*>[,<*F*>]

Execution (field)*(Rs + Displacement) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1	F	Rs				R	Rd			
Displacement															

Operands ***Rs(Displacement)**

Source register with displacement. The source operand location is the memory address specified by the sum of the specified register contents and the signed 16-bit displacement. The source displacement is contained in the extension word following the instruction.

F is an optional operand; it defaults to 0.
F=0 selects the FS0, FE0 parameters for the move.
F=1 selects the FS1, FE1 parameters for the move.

Description MOVE moves a field from the source memory address to the destination register. The source memory address is a bit address and is formed by adding the contents of the specified register to the signed 16-bit displacement. The field size for the above is 1–32 bits. When the field is moved into the destination register, it is right justified and sign extended or zero extended to 32 bits, according to the value of FE for the particular F bit selected. This instruction also performs an implicit compare to 0 of the field data. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 2

Machine States See Section 13.2, MOVE and MOVB Instructions Timing.

Status Bits
N *1* if the field-extended data moved to register is negative, *0* otherwise.
C Unaffected
Z *1* if the field-extended data moved to register is 0, *0* otherwise.
V *0*

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>15530	>3333
>15540	>4444
>15550	>5555

Code	Before			After	
	A0	FS0/1	FE0/1	A1	NCZV
MOVE *A0(>0000),A1,1	>0001 5530	x/1	x/1	>FFFF FFFF	1x00
MOVE *A0(>0003),A1,1	>0001 552F	x/2	x/0	>0000 0000	0x10
MOVE *A0(>0001),A1,0	>0001 552F	5/x	0/x	>0000 0013	0x00
MOVE *A0(>000F),A1,0	>0001 552D	8/x	1/x	>0000 0043	0x00
MOVE *A0(>0020),A1,1	>0001 551C	x/13	x/0	>0000 0443	0x00
MOVE *A0(>00FF),A1,0	>0001 5435	16/x	1/x	>0000 4333	0x00
MOVE *A0(>0FFF),A1,0	>0001 4531	19/x	1/x	>FFFC 3333	1x00
MOVE *A0(>7FFF),A1,1	>0000 D531	x/22	x/1	>0004 3333	0x00
MOVE *A0(>FFF2),A1,1	>0001 5540	x/25	x/0	>0111 0CCC	0x00
MOVE *A0(>8000),A1,0	>0001 D530	27/x	1/x	>FC44 3333	1x00
MOVE *A0(>FFF0),A1,0	>0001 5540	31/x	0/x	>4444 3333	0x00
MOVE *A0(>FFE0),A1,1	>0001 5558	x/31	x/1	>D544 4433	1x00
MOVE *A0(>FFEC),A1,0	>0001 554D	32/x	0/x	>AAA2 2219	1x00
MOVE *A0(>001D),A1,0	>0001 5520	32/x	1/x	>AAAA 2221	1x00
MOVE *A0(>0020),A1,1	>0001 5520	x/32	x/0	>5555 4444	0x00

Move Field - *Indirect with Displacement to Indirect (Postincrement)*

MOVE

Syntax **MOVE** ******<Rs(Displacement)>*, ******<Rd>*+[,*<F>*]

Execution
(field)*Rs(Displacement) → (field)*Rd
(Rd) + field size → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	0	F	Rs				R	Rd			
Displacement															

Operands

***Rs(Displacement)**
Source register with displacement. The source operand location is the memory address specified by the sum of the source register contents and the signed 16-bit displacement, contained in the extension word following the instruction.

***Rd+**
Destination register (indirect with postincrement). The destination location is the memory address contained in the specified register.

F is an optional operand; it defaults to 0.
F=0 selects the FS0 parameter for the move
F=1 selects the FS1 parameter for the move.

Description MOVE moves a field from the source memory address to the destination memory address contained in the destination register; both the source and destination memory addresses are bit addresses. The source memory address is formed by adding the contents of the source register to the signed 16-bit displacement. The destination register is incremented following the move by the field size selected. The field size for the move is 1–32 bits. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 2

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data	Address	Data
>15500	>0000	>15530	>3333
>15510	>0000	>15540	>4444
>15520	>0000	>15550	>5555

Code	Before A0	Before A1	FS0/1	After A1	@>15500	@>15510	@>15520
MOVE *A0(>0000),A1+,1	>00015530	>00015500	x/1	>00015501	>0001	>0000	>0000
MOVE *A0(>0001),A1+,1	>0001552F	>00015504	5/x	>00015509	>0130	>0000	>0000
MOVE *A0(>000F),A1+,1	>0001552D	>0001550C	8/x	>00015514	>3000	>0004	>0000
MOVE *A0(>0020),A1+,1	>0001551C	>0001550D	x/13	>0001551A	>6000	>0088	>0000
MOVE *A0(>00FF),A1+,1	>00015535	>0001550C	16/x	>0001551C	>3000	>0433	>0000
MOVE *A0(>0FFF),A1+,1	>00015531	>00015510	19/x	>00015523	>0000	>3333	>0004
MOVE *A0(>7FFF),A1+,1	>0000D531	>00015508	x/22	>0001551E	>3300	>0433	>0000
MOVE *A0(>FFF2),A1+,1	>00015540	>00015500	x/25	>00015519	>0CCC	>0111	>0000
MOVE *A0(>8000),A1+,1	>0001D530	>00015503	27/x	>0001551E	>9998	>2221	>0000
MOVE *A0(>FFF0),A1+,1	>00015540	>00015501	31/x	>0001552A	>6666	>8888	>0000
MOVE *A0(>FFE0),A1+,1	>00015558	>00015508	x/31	>00015527	>3300	>4444	>0055
MOVE *A0(>FFEC),A1+,1	>0001554D	>0001550A	32/x	>00015528	>3200	>4444	>0155
MOVE *A0(>001D),A1+,1	>00015520	>00015510	32/x	>00015530	>0000	>2221	>AAAA
MOVE *A0(>0020),A1+,1	>00015520	>00015510	x/32	>00015530	>0000	>4444	>5555

Syntax **MOVE** *<Rs(Displacement)>, *<Rd>(Displacement)>[,<F>]

Execution (field)*Rs(Displacement) → (field)*Rd(Displacement)

Encoding

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>F</td><td colspan="4">Rs</td><td>R</td><td colspan="4">Rd</td></tr>
<tr><td colspan="16">Source Displacement</td></tr>
<tr><td colspan="16">Destination Displacement</td></tr>
</table>

Operands

***Rs(Displacement)**

Source register with displacement. The source operand location is the memory address specified by the sum of the specified register contents and the signed 16-bit displacement, contained in the first of two extension words following the instruction.

***Rd(Displacement)**

Destination register with displacement. The destination location is the memory address specified by the sum of the specified register contents and the signed 16-bit displacement, contained in the second of two extension words following the instruction.

F is an optional operand; it defaults to 0.
F=0 selects the FS0 parameter for the move.
F=1 selects the FS1 parameter for the move.

Description MOVE moves a field from the source memory address to the destination memory address. Both the source and destination memory addresses are bit addresses and are formed by adding the contents of the specified register to its respective signed 16-bit displacement. The field size for the move is 1–32 bits. The SETF instruction sets the field size and extension. The source and destination registers must be in the same register file.

Words 3

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data	Address	Data
>15500	>0000	>15530	>3333
>15510	>0000	>15540	>4444
>15520	>0000	>15550	>5555

Before	After A0	A1	FS0/1	@>15500	@>15510	@>15520
`MOVE *A0(>0000),*A1(>0000),1`	>00015530	>00015500	x/1	>0001	>0000	>0000
`MOVE *A0(>0001),*A1(>0000),0`	>0001552F	>00015504	5/x	>0130	>0000	>0000
`MOVE *A0(>000F),*A1(>000F),0`	>0001552D	>000154FD	8/x	>3000	>0004	>0000
`MOVE *A0(>0020),*A1(>001D),1`	>0001551C	>000154F0	x/13	>6000	>0088	>0000
`MOVE *A0(>00FF),*A1(>FFF8),0`	>00015435	>00015514	16/x	>3000	>0433	>0000
`MOVE *A0(>0FFF),*A1(>0FFF),0`	>00014531	>00014511	19/x	>0000	>3333	>0004
`MOVE *A0(>7FFF),*A1(>8000),1`	>0000D531	>0001D508	x/22	>3300	>0433	>0000
`MOVE *A0(>FFF2),*A1(>7FFF),1`	>00015540	>0000D501	x/25	>0CCC	>0111	>0000
`MOVE *A0(>8000),*A1(>0020),0`	>0001D530	>000154E3	27/x	>9998	>2221	>0000
`MOVE *A0(>FFF0),*A1(>0010),0`	>00015540	>000154F1	31/x	>6666	>8888	>0000
`MOVE *A0(>FFE0),*A1(>FFE0),1`	>00015558	>00015528	x/31	>3300	>4444	>0055
`MOVE *A0(>FFEC),*A1(>FFEC),0`	>0001554D	>0001551D	32/x	>3200	>4444	>0155
`MOVE *A0(>001D),*A1(>0020),0`	>00015520	>000154F0	32/x	>0000	>2221	>AAAA
`MOVE *A0(>0020),*A1(>0020),1`	>00015520	>000154F0	x/32	>0000	>4444	>5555

Syntax **MOVE** *@<SAddress>,<Rd>*[,< *F*>]

Execution (field)@SAddress → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	F	1	1	0	1	R	Rd			
Source Address (LSW)															
Source Address (MSW)															

Operands **SAddress**

Source address. The source operand location is the linear memory address contained in the two extension words following the instruction. It is 1-32 bits in size.

F is an optional operand; it defaults to 0.
F=0 selects the FS0, FE0 parameters for the move.
F=1 selects the FS1, FE1 parameters for the move.

Description MOVE moves a field from the source memory address to the destination register. The specified source memory address is a bit address and the field size for the move is 1-32 bits. When the field is moved into the destination register, it is right justified and sign extended or zero extended to 32 bits according to the value of FE for the particular F bit selected. This instruction also performs an implicit compare to 0 of the field data. The SETF instruction sets the field size and extension.

Words 3

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N *1* if the field-extended data moved to register is negative, *0* otherwise.
C Unaffected
Z *1* if the field-extended data moved to register is 0, *0* otherwise.
V *0*

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>15500	>7770
>15510	>7777

Code	Before		After	
	FE0/1	FS0/1	A1	NCZV
MOVE @>15500,A1,1	x/0	x/1	>0000 0000	0x10
MOVE @>15500,A1,0	0/x	5/x	>0000 0010	0x00
MOVE @>15503,A1,1	x/1	x/5	>0000 000E	0x00
MOVE @>15500,A1,0	0/x	12/x	>0000 0770	0x00
MOVE @>1550D,A1,1	x/1	x/12	>FFFF FBBB	1x00
MOVE @>15504,A1,0	1/x	18/x	>FFFF 7777	1x00
MOVE @>15500,A1,1	x/0	x/18	>0003 7770	0x00
MOVE @>15500,A1,0	0/x	27/x	>0777 7770	0x00
MOVE @>15500,A1,1	x/1	x/27	>FF77 7770	1x00
MOVE @>15501,A1,0	0/x	30/x	>3BBB BBB8	0x00
MOVE @>15501,A1,1	x/1	x/30	>FBBB BBB8	1x00
MOVE @>15500,A1,0	x/x	32/x	>7777 7770	0x00

Syntax **MOVE** @<*SAddress*>, *<*Rd*>+[,F]

Execution (field)@SAddress → (field)*Rd
(Rd) + field size → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	F	0	0	0	0	R	Rd			
Source Address (LSW)															
Source Address (MSW)															

Operands **SAddress**

Source address. The source operand location is the linear memory address contained in the two extension words following the instruction.

***Rd+**

Destination register (indirect with postincrement). The destination location is the memory address contained in the specified register.

F is an optional operand; it defaults to 0.
F=0 selects the FS0 parameter for the move.
F=1 selects the FS1 parameter for the move.

Description MOVE moves a field from the source memory address to the memory address contained in the destination register. The source memory address is contained in the two extension words following the instruction. The destination register is incremented following the move by the field size selected. The source and destination registers must be in the same register file.

Words 5

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data	Address	Data
>15500	>FFFF	>15530	>0000
>15510	>FFFF	>15540	>0000
>15520	>FFFF	>15550	>0000

Code	Before A0	Before A1	After FE0/1	After A1	After @>15500	After @>15510	After @>15520
MOVE @15500,A1+,1	>00015530	>00015531	x/1	>00015531	>0001	>0000	>0000
MOVE @15500,A1+,0	>00015534	>00015539	5/x	>00015539	>01F0	>0000	>0000
MOVE @15500,A1+,1	>0001553A	>00015544	x/10	>00015544	>FC00	>000F	>0000
MOVE @15500,A1+,0	>0001553F	>00015552	19/x	>00015552	>8000	>FFFF	>0003
MOVE @15504,A1+,1	>00015530	>00015537	x/7	>00015537	>007F	>0000	>0000
MOVE @1550A,A1+,0	>00015530	>0001553D	13/x	>0001553D	>1FFF	>0000	>0000
MOVE @1550D,A1+,1	>00015534	>00015536	x/8	>00015536	>0FF0	>0000	>0000
MOVE @1550D,A1+,0	>00015530	>0001554C	28/x	>0001554C	>FFFF	>0FFF	>0000
MOVE @15505,A1+,1	>00015535	>0001554D	x/23	>0001554D	>FFE0	>0FFF	>0000
MOVE @15508,A1+,0	>00015536	>00015555	31/x	>00015555	>FFC0	>FFFF	>001F
MOVE @15508,A1+,1	>00015531	>00015548	x/31	>00015548	>FFFE	>FFFF	>0000
MOVE @1550A,A1+,0	>00015530	>00015550	32/x	>00015550	>FFFF	>FFFF	>0000
MOVE @15500,A1+,1	>0001553A	>0001555A	x/32	>0001555A	>FC00	>FFFF	>03FF

Syntax MOVE *@<SAddress>, @<DAddress>*[,*<F>*]

Execution (field)@SAddress → (field)@DAddress

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	F	1	1	1	0	0	0	0	0	0
Source Address (LSW)															
Source Address (MSW)															
Destination Address (LSW)															
Destination Address (MSW)															

Operands

SAddress
Source address. The source operand location is the linear memory address contained in the first set of two extension words following the instruction.

DAddress
Destination address. The destination location is the linear memory address contained in the second set of two extension words following the instruction.

F is an optional operand; it defaults to 0.
F=0 selects the FS0 parameter for the move.
F=1 selects the FS1 parameter for the move.

Description MOVE moves a field from the source memory address to the destination memory address. Both memory addresses are bit addresses and the field size for the move is 1–32 bits. The SETF instruction sets the field size and extension.

Words 5

Machine States See MOVE and MOVB Instructions Timing, Section 13.2.

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Assume that memory contains the following values before instruction execution:

Address	Data
>15500	>FFFF
>15510	>FFFF
>15520	>FFFF
>15530	>0000
>15540	>0000
>15550	>0000

Code	Before	After		
	FS0/1	@>15530	@>15540	@>15550
`MOVE @>15500,@>15530,1`	x/1	>0001	>0000	>0000
`MOVE @>15500,@>15534,0`	5/x	>01F0	>0000	>0000
`MOVE @>15500,@>1553A,1`	x/10	>FC00	>000F	>0000
`MOVE @>15500,@>1553F,0`	19/x	>8000	>FFFF	>0003
`MOVE @>15504,@>15530,1`	x/7	>007F	>0000	>0000
`MOVE @>1550A,@>15530,0`	13/x	>1FFF	>0000	>0000
`MOVE @>1550D,@>15534,1`	x/8	>0FF0	>0000	>0000
`MOVE @>1550D,@>15530,0`	28/x	>FFFF	>0FFF	>0000
`MOVE @>15505,@>15535,1`	x/23	>FFE0	>0FFF	>0000
`MOVE @>15508,@>15536,0`	31/x	>FFC0	>FFFF	>001F
`MOVE @>15508,@>15531,1`	x/31	>FFFE	>FFFF	>0000
`MOVE @>1550A,@>15530,0`	32/x	>FFFF	>FFFF	>0000
`MOVE @>15500,@>1553A,0`	x/32	>FC00	>FFFF	>03FF

Syntax **MOVI** *<IW>*,*<Rd>*[,**W**]

Execution IW → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	1	0	R	Rd			
IW															

Operands **IW** is a 16-bit immediate value.

Description MOVI stores a 16-bit, sign-extended immediate value in the destination register.

The assembler will use the short form if the immediate value has been previously defined and is in the range -32,768 ≤ IW ≤ 32,767. You can force the assembler to use the short form by following the register specification with **,W**:

```
MOVI IW,Rd,W
```

The assembler will truncate the upper bits and issue an appropriate warning message.

Words 2

Machine States 2,8

Status Bits
- **N** *1* if the data being moved is negative, *0* otherwise.
- **C** Unaffected
- **Z** *1* if the data being moved is 0, *0* otherwise.
- **V** *0*

Examples

Code	After A0	NCZV
`MOVI 32767,A0`	>0000 7FFF	0x00
`MOVI 1,A0`	>0000 0001	0x00
`MOVI 0,A0`	>0000 0000	0x10
`MOVI -1,A0`	>FFFF FFFF	1x00
`MOVI -32768,A0`	>FFFF 8000	1x00
`MOVI >0000,A0`	>0000 0000	0x10
`MOVI >7FFF,A0`	>0000 7FFF	0x00

Syntax **MOVI** <*IL*>,<*Rd*>[,**L**]

Execution IL → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	1	1	R	Rd			
IL (LSW)															
IL (MSW)															

Operands **IL** is a 32-bit immediate value.

Description MOVI stores a 32-bit immediate value in the destination register. The assembler will use this opcode if it cannot use the `MOVI IW,Rd` opcode, or if the long opcode is forced by following the register specification with ,**L**:

```
MOVI IL,Rd,L
```

Words 3

Machine States 3,12

Status Bits
N *1* if the data being moved is negative, *0* otherwise.
C Unaffected
Z *1* if the data being moved is 0, *0* otherwise.
V *0*

Examples

Code	After A0	NCZV
`MOVI 2147483647,A0`	>7FFF FFFF	0x00
`MOVI 32768,A0`	>0000 8000	0x00
`MOVI -32769,A0`	>FFFF 7FFF	1x00
`MOVI -2147483648,A0`	>8000 0000	1x00
`MOVI >8000,A0`	>0000 8000	0x00
`MOVI >FFFFFFFF,A0`	>FFFF FFFF	1x00
`MOVI >FFFF,A0,L`	>FFFF FFFF	1x00

MOVK Move Constant (5 Bits)

Syntax **MOVK** *<K>,<Rd>*

Execution K → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0			K			R		Rd		

Operands **K** is a constant from 1 to 32.

Description MOVK stores a 5-bit constant in the destination register. The constant is treated as an unsigned number in the range 1–32, where K = 0 in the opcode corresponds to a value of 32. The resulting constant value is zero extended to 32 bits. Note that you cannot set a register to 0 with this instruction. You can clear a register by XORing the register with itself; use `CLR Rd` (an alternate mnemonic for XOR) to accomplish this. Both these methods alter the Z bit (set it to 1).

Words 1

Machine States 1,4

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples

Code	After A0
`MOVK 1,A0`	>0000 0001
`MOVK 8,A0`	>0000 0008
`MOVK 16,A0`	>0000 0010
`MOVK 32,A0`	>0000 0020

Syntax **MOVX** *<Rs>,<Rd>*

Execution (Rs**X**) → Rd**X**

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	Rs				R	Rd			

Description MOVX moves the X half of the source register (16 LSBs) to the X half of the destination register. The Y halves of both registers are unaffected.

MOVX and MOVY instructions can be used for handling packed 16-bit quantities and XY addresses. The RL instruction can be used to swap the contents of X and Y.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before		After
	A0	**A1**	**A1**
`MOVX A0,A1`	>0000 0000	>FFFF FFFF	>FFFF 0000
`MOVX A0,A1`	>1234 5678	>0000 0000	>0000 5678
`MOVX A0,A1`	>FFFF FFFF	>0000 0000	>0000 FFFF

Syntax MOVY <*Rs*>,<*Rd*>

Execution (RsY) → RdY

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	Rs				R	Rd			

Description MOVY moves the Y half of the source register (16 MSBs) to the Y half of the destination register. The X halves of both registers are unaffected.

MOVX and MOVY instructions can be used for handling packed 16-bit quantities and XY addresses. The RL instruction can be used to swap the contents of X and Y.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples

Code	Before A0	Before A1	After A1
`MOVY A0,A1`	>0000 0000	>FFFF FFFF	>0000 FFFF
`MOVY A0,A1`	>1234 5678	>0000 0000	>1234 0000
`MOVY A0,A1`	>FFFF FFFF	>0000 0000	>FFFF 0000

Syntax **MPYS** *<Rs>,<Rd>*

Execution
Rd Even: (Rs) × (Rd) → Rd:Rd+1
Rd Odd: (Rs) × (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	Rs				R	Rd			

Description There are two cases:

Rd Even MPYS performs a signed multiply of the source register by the destination register, and stores the 64-bit result in the two consecutive registers starting at the destination register. The 32 MSBs of the result are stored in the specified even-numbered destination register. The 32 LSBs of the result are stored in the next consecutive register, which is odd-numbered. Avoid using A14 or B14 as the destination register, since this overwrite the SP. The assembler will issue a warning in this case.

Rd Odd Perform a signed multiply of the source register by the destination register, and store the 32 LSBs of the result in the destination register. Note that overflows are not detected. The Z and N bits are set on the full 64-bit result, even though only the lower 32 bits are stored in Rd.

FS1 controls the width of the multiply; the portion of Rs by which Rd is multiplied is determined by FS1. FS1 should be even. If FS1 is odd, MPYS will produce unpredictable results. The MSB of the source operand field supplies the source operand's sign. The source and destination registers must be in the same register file.

Words 1

Machine States 20,23

Status Bits
N *1* if the result is negative, *0* otherwise.
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples `MPYS A1, A0`

Before A0	Before A1	FS1	After A0	After A1	NCZV
>0000 0000	>0000 0000	32	>0000 0000	>0000 0000	0x1x
>0000 0000	>7FFF FFFF	32	>0000 0000	>0000 0000	0x1x
>0000 0000	>FFFF FFFF	32	>0000 0000	>0000 0000	0x1x
>7FFF FFFF	>0000 0000	32	>0000 0000	>0000 0000	0x1x
>FFFF FFFF	>0000 0000	32	>0000 0000	>0000 0000	0x1x
>7FFF 0000	>1000 0000	32	>0000 0000	>7FFF 0000	0x0x
>7FFF 0000	>1000 0000	32	>0000 007F	>FF00 0000	0x0x
>7FFF 0000	>1000 0000	32	>0000 7FFF	>0000 0000	0x0x
>FFFF FFFF	>1000 0000	32	>FFFF FFFF	>FFFF FFFF	1x0x
>8000 0000	>7FFF FFFF	32	>C000 0000	>8000 0000	1x0x
>FFFF 0000	>7FFF 0000	32	>FFFF 8001	>0000 0000	1x0x
>FFFF FFFF	>FFFF FFFF	32	>0000 0000	>1000 0000	0x0x
>8000 0000	>8000 0000	32	>4000 0000	>0000 0000	0x0x
>8000 0001	>8000 0000	32	>3FFF FFFF	>8000 0000	0x0x

`MPYS A0,A1`

Before A0	Before A1	FS1	After A0	After A1	NCZV
>0000 0000	>0000 0000	32	>0000 0000	>0000 0000	0x1x
>FFFF FFFF	>0000 0000	32	>FFFF FFFF	>0000 0000	0x1x
>0000 0000	>7FFF FFFF	32	>0000 0000	>0000 0000	0x1x
>7FFF 0000	>1000 0000	32	>007F FF00	>7FFF 0000	0x0x
>7FFF 0000	>1000 0000	32	>007F FF00	>FF00 0000	0x0x
>7FFF 0000	>1000 0000	32	>007F FF00	>0000 0000	0x0x
>FFFF FFFF	>1000 0000	32	>FFFF FFFF	>FFFF FFFF	1x0x
>FFFF 0000	>7FFF 0000	32	>FFFF 0000	>0000 0000	1x0x
>FFFF FFFF	>FFFF FFFF	32	>FFFF FFFF	>1000 0000	0x0x
>8000 0001	>8000 0000	32	>8000 0001	>8000 0000	0x0x
>8000 0000	>8000 0000	32	>8000 0000	>0000 0000	0x1x

Syntax **MPYU** *<Rs>,<Rd>*

Execution
Rd Even: (Rs) × (Rd) → Rd:Rd+1
Rd Odd: (Rs) × (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	Rs				R	Rd			

Description There are two cases:

Rd Even MPYU performs an unsigned multiply of the source register by the destination register, and stores the 64-bit result in the two consecutive registers starting at the destination register. The 32 MSBs of the result are stored in the specified even-numbered destination register. The 32 LSBs of the result are stored in the next consecutive register, which is odd-numbered. Avoid using A14 or B14 as the destination register, since this overwrites the SP. The assembler will issue a warning in this case.

Rd Odd Perform an unsigned multiply of the source register by the destination register, and store the 32 LSBs of the result in the destination register. Note that overflows are not detected. The Z and N bits are set on the full 64-bit result, even though only the lower 32 bits are stored in Rd.

FS1 controls the width of the multiply; the portion of Rs by which Rd is multiplied is determined by FS1. FS1 should be even. If FS1 is odd, MPYS will produce unpredictable results.

The source and destination registers must be in the same register file.

Words 1

Machine States 21,24

Status Bits
N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples `MPYU A1,A0`

Before A0	Before A1	FS1	After A0	After A1	NCZV
>0000 0000	>0000 0000	32	>0000 0000	>0000 0000	xx1x
>0000 0000	>FFFF FFFF	32	>0000 0000	>0000 0000	xx1x
>FFFF FFFF	>0000 0000	32	>0000 0000	>0000 0000	1x1x
>FFFF 0000	>1000 0000	32	>0000 0000	>FFFF 0000	xx0x
>FFFF 0000	>1000 0000	32	>0000 00FF	>FF00 0000	xx0x
>FFFF 0000	>1000 0000	32	>0000 FFFF	>0000 0000	xx0x

```
MPYU A0,A1
```

Before			After		
A0	**A1**	**FS1**	**A0**	**A1**	**NCZV**
>0000 0000	>0000 0000	32	>0000 0000	>0000 0000	xx1x
>FFFF FFFF	>0000 0000	32	>FFFF FFFF	>0000 0000	xx1x
>0000 0000	>FFFF FFFF	32	>0000 0000	>0000 0000	1x1x
>FFFF 0000	>1000 0000	32	>00FF FF00	>FFFF 0000	xx0x
>FFFF 0000	>1000 0000	32	>00FF FF00	>FF00 0000	xx0x
>FFFF 0000	>1000 0000	32	>00FF FF00	>0000 0000	xx0x

Syntax NEG *<Rd>*

Execution -(Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	0	1	R	Rd			

Description NEG stores the 2's complement of the contents of the destination register back into the destination register.

Words 1

Machine States 1,4

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow (Rd ≠ 0), *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow (Rd = >8000 0000), *0* otherwise.

Examples

Code	Before	After	
	A0	NCZV	A0
NEG A0	>0000 0000	0010	>0000 0000
NEG A0	>5555 5555	1100	>AAAA AAAB
NEG A0	>7FFF FFFF	1100	>8000 0001
NEG A0	>8000 0000	1101	>8000 0000
NEG A0	>8000 0001	0100	>7FFF FFFF
NEG A0	>FFFF FFFF	0100	>0000 0001

NEGB — Negate Register with Borrow — NEGB

Syntax NEGB *<Rd>*

Execution -(Rd) - (C) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	0	R	Rd			

Description NEGB takes the 2's complement of the destination register's contents and decrements by 1 if the borrow bit (C) is set; the result is stored in the destination register. This instruction can be used in sequence with itself and with the NEG instruction for negating multiple-register quantities.

Words 1

Machine States 1,4

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A0	Before C	After NCZV	After A0
NEGB A0	>0000 0000	0	0010	>0000 0000
NEGB A0	>0000 0000	1	1100	>FFFF FFFF
NEGB A0	>5555 5555	0	1100	>AAAA AAAB
NEGB A0	>5555 5555	1	1100	>AAAA AAAA
NEGB A0	>7FFF FFFF	0	1100	>8000 0001
NEGB A0	>7FFF FFFF	1	1100	>8000 0000
NEGB A0	>8000 0000	0	1101	>8000 0000
NEGB A0	>8000 0000	1	0100	>7FFF FFFF
NEGB A0	>8000 0001	0	0100	>7FFF FFFF
NEGB A0	>8000 0001	1	0100	>7FFF FFFE
NEGB A0	>FFFF FFFF	0	0100	>0000 0001
NEGB A0	>FFFF FFFF	1	0110	>0000 0000

Syntax NOP

Execution No operation

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Description The program counter is incremented to point to the next instruction. The processor status is otherwise unaffected.

This instruction can be used to pad loops and perform other timing functions.

Words 1

Machine States 1,4

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Example

Code	Before	After
	PC	PC
NOP	>00020000	>00020010

Syntax **NOT** *<Rd>*

Execution NOT(Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	1	R	Rd			

Description NOT stores the 1's complement of the destination register's contents back into the destination register.

Words 1

Machine States 1,4

Status Bits
N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before	After	
	A0	NCZV	A0
NOT A0	>0000 0000	xx0x	>FFFF FFFF
NOT A0	>5555 5555	xx0x	>AAAA AAAA
NOT A0	>FFFF FFFF	xx1x	>0000 0000
NOT A0	>8000 0000	xx0x	>7FFF FFFF

OR OR Registers OR

Syntax **OR** *<Rs>,<Rd>*

Execution (Rs) OR (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	Rs				R	Rd			

Description This instruction bitwise-ORs the contents of the source register with the contents of the destination register; the result is stored in the destination register.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before A0	Before A1	After A1	NCZV
`OR A0,A1`	>FFFF FFFF	>0000 0000	>FFFF FFFF	xx0x
`OR A0,A1`	>0000 0000	>FFFF FFFF	>FFFF FFFF	xx0x
`OR A0,A1`	>5555 5555	>AAAA AAAA	>FFFF FFFF	xx0x
`OR A0,A1`	>0000 0000	>0000 0000	>0000 0000	xx1x

Syntax **ORI** *<L>,<Rd>*

Execution L OR (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	0	1	R	Rd			
L (LSW)															
L (MSW)															

Operands **L** is a 32-bit immediate value.

Description This instruction bitwise-ORs the 32-bit immediate value, L, with the contents of the destination register; the result is stored in the destination register.

Words 3

Machine States 3,12

Status Bits
N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before A0	After A0	NCZV
`ORI >FFFFFFFF,A0`	>0000 0000	>FFFF FFFF	xx0x
`ORI >00000000,A0`	>FFFF FFFF	>FFFF FFFF	xx0x
`ORI >AAAAAAAA,A0`	>5555 5555	>FFFF FFFF	xx0x
`ORI >00000000,A0`	>0000 0000	>0000 0000	xx1x

Syntax **PIXBLT** B,L

Execution Binary source pixel array → Destination pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0

Operands

B specifies that the source pixel array is treated as a binary array whose starting address is given in linear format.

L specifies that the destination pixel array starting address is given in linear format.

Description PIXBLT expands, transfers, and processes a binary source pixel array with a destination pixel array. This instruction operates on two-dimensional arrays of pixels using linear starting addresses for both the source and the destination. The source pixel array is treated as a one bit per pixel array. As the PixBlt proceeds, the source pixels are expanded and then combined with the corresponding destination pixels based on the selected graphics operations.

Note that the instruction is entered as `PIXBLT B,L`. The following set of implied operands govern the operation of the instruction and define the source and destination arrays.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B0†	SADDR	Linear	Source pixel array starting address
B1	SPTCH	Linear	Source pixel array pitch
B2†	DADDR	Linear	Destination pixel array starting address
B3	DPTCH	Linear	Destination pixel array pitch
B7	DYDX	XY	Pixel array dimensions (rows:columns)
B8	COLOR0	Pixel	Background expansion color
B9	COLOR1	Pixel	Foreground expansion color
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP– Pixel processing operations (22 options) T – Transparency operation
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† These registers are changed by PIXBLT execution.

Source Array The source pixel array for the expand operation is defined by the contents of the SADDR, SPTCH, and DYDX registers:

- At the outset of the instruction, SADDR contains the **linear** address of the pixel with the lowest address in the array.

- SPTCH contains the linear difference in the starting addresses of adjacent rows of the source array. SPTCH can be any pixel-aligned value for this PIXBLT.

- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of pixels per row.

During instruction execution, SADDR points to the address of the next set of 32 pixels to be read from the source array. When the transfer is complete, SADDR points to the linear address of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Source Expansion

The actual source pixel values which are to be written or processed with the destination array are determined by the interaction of the source array with the contents of the COLOR1 and COLOR0 registers. In the expansion operation, a **1** bit in the source array selects a pixel from the COLOR1 register for operation on the destination array. A **0** bit in the source array selects a COLOR0 pixel for this purpose. The pixels selected from the COLOR1 and COLOR0 registers are those that align directly with their intended position in the destination array word.

Destination Array

The location of the destination pixel block is defined by the contents of the DADDR, DPTCH, and DYDX registers:

- At the outset of the instruction, DADDR contains the **linear** address of the pixel with the lowest address in the array.

- DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array (typically this is the screen pitch). DPTCH **must** be a multiple of 16.

- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of pixels per row.

During instruction execution, DADDR points to the next pixel (or word of pixels) to be modified in the destination array. When the block transfer is complete, DADDR points to the linear address of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Corner Adjust

No corner adjust is performed for this instruction; PBH and PBV are ignored. The pixel transfer simply proceeds in the order of increasing linear addresses.

Window Checking

Window checking **cannot** be used with this PixBlt instruction. The contents of the WSTART and WEND registers are ignored.

Pixel Processing

Pixel processing can be used with this instruction. The PPOP field of the CONTROL I/O register specifies the pixel processing operation that will be applied to *expanded pixels* as they are processed with the destination array. There are 16 Boolean and 6 arithmetic operations; the default case at reset

is the *replace* (S → D) operation. Note that the data is *first expanded* and *then processed*. The 6 arithmetic operations do not operate with pixel sizes of one or two bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it expands and processes the source data. At reset, the default case for transparency is *off*.

Plane Mask The plane mask is enabled for this instruction.

Interrupts This instruction can be interrupted at a word or row boundary of the destination array. When the PixBlt is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values. DADDR points to the linear address of the next word of pixels to be modified after the interrupt is processed. SADDR points to the address of the next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the PixBlt correctly. You can inhibit the TMS34010 from resuming the PixBlt by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Shift Register Transfers If the SRT bit in the DPYCTL I/O register is set, each memory read or write initiated by the PixBlt generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory clears or transfers. (Not all VRAMs support this capability.)

Words 1

Machine States See PIXBLT Expand Instructions Timing, Section 13.5.

Status Bits
- **N** Undefined
- **C** Undefined
- **Z** Undefined
- **V** Undefined

Examples Before the PIXBLT instruction can be executed, the implied operand registers must be loaded with appropriate values. These PIXBLT examples use the following implied operand setup.

Register File B:

SADDR (B0) = >0000 2030
SPTCH (B1) = >0000 0100
DADDR (B2) = >0003 3000
DPTCH (B3) = >0000 1000
DYDX (B7) = >0002 0010
COLOR0 (B8) = >FEDC FEDC
COLOR1 (B9) = >BA98 BA98

I/O Registers:

PSIZE = >0010

For this example, assume that memory contains the following data before instruction execution.

Linear Address	Data
>02000	>xxxx, >xxxx, >xxxx, >1234, >xxxx, >xxxx, >xxxx, >xxxx
>02080	>xxxx, >xxxx, >xxxx, >xxxx, >xxxx, >xxxx, >xxxx, >xxxx
>02100	>xxxx, >xxxx, >xxxx, >5678, >xxxx, >xxxx, >xxxx, >xxxx
>02180	>xxxx, >xxxx, >xxxx, >xxxx, >xxxx, >xxxx, >xxxx, >xxxx
>33000	>FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF
>33080	>FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF
>34000	>FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF
>34080	>FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF

Example 1 This example uses the *replace (S → D)* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >0000 (T=0, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data
>33000	>FEDC,>FEDC,>BA98,>FEDC,>BA98,>BA98,>FEDC,>FEDC
>33080	>FEDC,>BA98,>FEDC,>FEDC,>BA98,>FEDC,>FEDC,>FEDC
>34000	>FEDC,>FEDC,>FEDC,>BA98,>BA98,>BA98,>BA98,>FEDC
>33080	>FEDC,>BA98,>BA98,>FEDC,>BA98,>FEDC,>BA98,>FEDC

Example 2 This example uses the *(D - S) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >4800 (T=0, PP=10010).

After instruction execution, memory will contain the following values:

Linear Address	Data
>33000	>0123, >0123, >4567, >0123, >4567, >4567, >0123, >0123
>33080	>0123, >4567, >0123, >0123, >4567, >0123, >0123, >0123
>34000	>0123, >0123, >0123, >4567, >4567, >4567, >4567, >0123
>34080	>0123, >4567, >4567, >0123, >4567, >0123, >4567, >0123

Example 3

This example uses transparency with COLOR0 = >00000000. Before instruction execution, PMASK = >0000 and CONTROL = >0020 (T=1, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data
>33000	>FFFF, >FFFF, >BA98, >FFFF, >BA98, >BA98, >FFFF, >FFFF
>33080	>FFFF, >BA98, >FFFF, >FFFF, >BA98, >FFFF, >FFFF, >FFFF
>34000	>FFFF, >FFFF, >FFFF, >BA98, >BA98, >BA98, >BA98, >FFFF
>34080	>FFFF, >BA98, >BA98, >FFFF, >BA98, >FFFF, >BA98, >FFFF

Example 4

This example uses plane masking; the four LSBs are masked. Before instruction execution, PMASK = >000F and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data
>33000	>FEDF, >FEDF, >BA9F, >FEDF, >BA9F, >BA9F, >FEDF, >FEDF
>33080	>FEDF, >BA9F, >FEDF, >FEDF, >BA9F, >FEDF, >FEDF, >FEDF
>34000	>FEDF, >FEDF, >FEDF, >BA9F, >BA9F, >BA9F, >BA9F, >FEDF
>34080	>FEDF, >BA9F, >BA9F, >FEDF, >BA9F, >FEDF, >BA9F, >FEDF

Syntax PIXBLT B,XY

Execution Binary source pixel array → Destination pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	1	0	1	0	0	0	0	0

Operands

B specifies that the source pixel array is treated as a binary array whose starting address is given in linear format.

XY specifies that the destination pixel array starting address is given in XY format.

Description PIXBLT expands, transfers, and processes a binary source pixel array with a destination pixel array. This instruction operates on two-dimensional arrays of pixels using a linear starting address for the source and an XY address for the destination. The source pixel array is treated as a one bit per pixel array. As the PixBlt proceeds, the source pixels are expanded and then combined with the corresponding destination pixels based on the selected graphics operations.

Note that the instruction is entered as `PIXBLT B,XY`. The following set of implied operands govern the operation of the instruction and define the source and destination arrays.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B0†	SADDR	Linear	Source pixel array starting address
B1	SPTCH	Linear	Source pixel array pitch
B2†‡	DADDR	XY	Destination pixel array starting address
B3	DPTCH	Linear	Destination pixel array pitch
B4	OFFSET	Linear	Screen origin (0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner
B7‡	DYDX	XY	Pixel array dimensions (rows:columns)
B8	COLOR0	Pixel	Background expansion color
B9	COLOR1	Pixel	Foreground expansion color
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP – Pixel processing operations (22 options) W – Window clipping or pick operation T – Transparency operation
>C0000130	CONVSP	XY-to-linear conversion (source pitch) Used for source preclipping.
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)
>C0000150	PSIZE	Pixel size (1,2,4,6,8,16)
>C0000160	PMASK	Plane mask – pixel format

† These registers are changed by PIXBLT execution.
‡ Used for common rectangle function with window hit operation (W=1).

Source Array The source pixel array for the expand operation is defined by the contents of the SADDR, SPTCH, DYDX, and (potentially) CONVSP registers:

- At the outset of the instruction, SADDR contains the **linear** address of the pixel with the lowest address in the array.
- SPTCH contains the linear difference in the starting addresses of adjacent rows of the source array. SPTCH can be any pixel-aligned value for this PIXBLT. For window clipping, SPTCH must be a power of two, and CONVSP must be set to correspond to the SPTCH value.
- CONVSP is computed by operating on the SPTCH register with the LMO instruction; it is used for the XY calculations involved in XY addressing and window clipping.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of pixels per row.

During instruction execution, SADDR points to the address of the next set of 32 pixels to be read from the source array. When the block transfer is complete, SADDR points to the linear address of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Source Expansion The actual source pixel values which are to be written or processed with the destination array are determined by the interaction of the source array with contents of the COLOR1 and COLOR0 registers. In the expansion operation, a **1** bit in the source array selects a pixel from the COLOR1 register for operation on the destination array. A **0** bit in the source array selects a COLOR0 pixel for this purpose. The pixels selected from the COLOR1 and COLOR0 registers are those that align directly with their intended position in the destination array word.

Destination Array The location of the destination pixel block is defined by the contents of the DADDR, DPTCH, CONVDP, OFFSET, and DYDX registers:

- At the outset of the instruction, DADDR contains the **XY** address of the pixel with the lowest address in the array. It is used with OFFSET and CONVDP to calculate the linear address of the array.
- DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array (typically this is the screen pitch). DPTCH **must** be a power of two (greater than or equal to 16) and CONVDP must be set to correspond to the DPTCH value.
- CONVDP is computed by operating on the DPTCH register with the LMO instruction; it is used for the XY calculations involved in XY addressing and window clipping.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of pixels per row.

During instruction execution, DADDR points to the **linear address** of next pixel (or word of pixels) to be modified in the destination array. When the block transfer is complete, DADDR points to the **linear address** of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Corner Adjust No corner adjust is performed for this instruction. The transfer executes in the order of increasing linear addresses. PBH and PBV are ignored.

Window Checking Window checking can be used with this instruction by setting the two W bits in the CONTROL register to the desired value. If window checking mode 1, 2, or 3 is selected, the WSTART and WEND registers define the XY starting and ending corners of a rectangular window.

0 *No windowing.* The entire pixel array is drawn and the WVP and V bits are unaffected.

1 *Window hit.* No pixels are drawn. The V bit is set to 0 if any portion of the destination array lies within the window. Otherwise, the V bit is set to 1.

If the V bit is set to 0, the DADDR and DYDX registers are modified to correspond to the common rectangle formed by the intersection of the destination array with the rectangular window. DADDR is set to the XY address of the pixel in the starting corner of the common rectangle. DYDX is set to the X and Y dimensions of the common rectangle.

If the V bit is set to 1, the array lies entirely outside the window, and the values of DADDR and DYDX are indeterminate.

2 *Window miss.* If the array lies **entirely** within the active window, it is drawn and the V bit is set to 0. Otherwise, no pixels are drawn, the V and WVP bits are set to 1, and the instruction is aborted.

3 *Window clip.* The source and destination arrays are preclipped to the window dimensions. Only those pixels that lie within the common rectangle (corresponding to the intersection of the specified array and the window) are drawn. If any preclipping is required, the V bit is set to 1.

Pixel Processing Pixel processing can be used with this instruction. The PPOP field of the CONTROL I/O register specifies the pixel processing operation that will be applied to *expanded pixels* as they are processed with the destination array. There are 16 Boolean and 6 arithmetic operations; the default case at reset is the S → D operation. Note that the data is *first expanded* and *then processed*. The 6 arithmetic operations do not operate with pixel sizes of one or two bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it expands and processes the source data. At reset, the default case for transparency is *off*.

Plane Mask The plane mask is enabled for this instruction.

Interrupts This instruction can be interrupted at a word or row boundary of the destination array. When the PixBlt is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values. DADDR points to the linear address of the next word of pixels to be modified after the interrupt is processed. SADDR points to the address of the next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the PixBlt correctly. You can inhibit the TMS34010 from resuming the PixBlt by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Shift Register Transfers If the SRT bit in the DPYCTL I/O register is set, each memory read or write initiated by the PixBlt generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory clears or transfers. (Not all VRAMs support this capability.)

Words 1

Machine States See PIXBLT Expand Instructions Timing, Section 13.5.

Status Bits

- **N** Undefined
- **C** Undefined
- **Z** Undefined
- **V** *1* if a window violation occurs, *0* otherwise. Undefined if window checking is not enabled (W=00).

Examples

Before the PIXBLT instruction can be executed, the implied operand registers must be loaded with appropriate values. These PIXBLT examples use the following implied operand setup.

Register File B:
SADDR (B0) = >0000 2010
SPTCH (B1) = >0000 0010
DADDR (B2) = >0030 0022
DPTCH (B3) = >0000 1000
OFFSET (B4) = >0001 0000
WSTART (B5) = >0000 0026
WEND (B6) = >0040 0050
DYDX (B7) = >0004 0010
COLOR0 (B8) = >0000 0000
COLOR1 (B9) = >7C7C 7C7C

I/O Registers:
PSIZE = >0008

Additional implied operand values are listed with each example.

For this example, assume that memory contains the following data before instruction execution.

Linear Address	Data
>2000	>xxxx, >0123, >4567, >89AB, >CDEF, >xxxx, >xxxx, >xxxx
>40000 to >43080	>FFFF

Example 1

This example uses the *replace (S → D)* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data
>40100	>FFFF, >7C7C, >0000, >7C00, >0000, >007C, >0000, >0000
>40180	>0000, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF
>41100	>FFFF, >7C7C, >007C, >7C00, >007C, >007C, >007C, >0000
>41180	>007C, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF
>42100	>FFFF, >7C7C, >7C00, >7C00, >7C00, >007C, >7C00, >0000
>42180	>7C00, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF
>43100	>FFFF, >7C7C, >7C7C, >7C00, >7C7C, >007C, >7C7C, >0000
>43180	>7C7C, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF, >FFFF

XY Addressing

X Address

Y Address	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34
30	FF	FF	7C	7C	00	00	00	7C	00	00	7C	00	00	00	00	00	00	00	FF	FF	FF
31	FF	FF	7C	7C	7C	00	00	7C	7C	00	7C	00	7C	00	00	00	7C	00	FF	FF	FF
32	FF	FF	7C	7C	00	7C	00	7C	00	7C	7C	00	00	7C	00	00	00	7C	FF	FF	FF
33	FF	FF	7C	7C	7C	7C	00	7C	7C	7C	7C	00	7C	7C	00	00	7C	7C	FF	FF	FF

Example 2

This example uses the XOR pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >2800 (T=0, W=00, PP=01010).

After instruction execution, memory will contain the following values:

X Address

Y Address	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34
30	FF	FF	83	83	FF	FF	FF	83	FF	FF	83	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
31	FF	FF	83	83	83	FF	FF	83	83	FF	83	FF	83	FF	FF	FF	83	FF	FF	FF	FF
32	FF	FF	83	83	FF	83	FF	83	FF	83	83	FF	FF	83	FF	FF	FF	83	FF	FF	FF
33	FF	FF	83	83	83	83	FF	83	83	83	83	FF	83	83	FF	FF	83	83	FF	FF	FF

Example 3

This example uses transparency. Before instruction execution, PMASK = >0000 and CONTROL = >0020 (T=1, W=00, PP=00000).

After instruction execution, memory will contain the following values:

X Address

Y Address	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34
30	FF	FF	7C	7C	FF	FF	FF	7C	FF	FF	7C	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
31	FF	FF	7C	7C	7C	FF	FF	7C	7C	FF	7C	FF	7C	FF	FF	FF	7C	FF	FF	FF	FF
32	FF	FF	7C	7C	FF	7C	FF	7C	FF	7C	7C	FF	FF	7C	FF	FF	FF	7C	FF	FF	FF
33	FF	FF	7C	7C	7C	7C	FF	7C	7C	7C	7C	FF	7C	7C	FF	FF	7C	7C	FF	FF	FF

Example 4 This example uses window operation 3 (clipped destination). Before instruction execution, PMASK = >0000 and CONTROL = >00C0 (T=0, W=11, PP=00000).

After instruction execution, memory will contain the following values:

X Address

Y Address	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34
30	FF	FF	FF	FF	FF	FF	00	7C	00	00	7C	00	00	00	00	00	00	00	FF	FF	FF
31	FF	FF	FF	FF	FF	FF	00	7C	7C	00	7C	00	7C	00	00	00	7C	00	FF	FF	FF
32	FF	FF	FF	FF	FF	FF	00	7C	00	7C	7C	00	00	7C	00	00	00	7C	FF	FF	FF
33	FF	FF	FF	FF	FF	FF	00	7C	7C	7C	7C	00	7C	7C	00	00	7C	7C	FF	FF	FF

Example 5 This example uses plane masking; the four LSBs of each pixel are masked. Before instruction execution, PMASK = >0F0F and CONTROL = >0020 (T=1, W=00, PP=00000).

After instruction execution, memory will contain the following values:

X Address

Y Address	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34
30	FF	FF	FF	FF	FF	FF	FF	7F	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
31	FF	FF	FF	FF	FF	FF	FF	7F	7F	FF	FF	FF	FF	FF	FF	FF	7F	FF	FF	FF	FF
32	FF	FF	FF	FF	FF	FF	FF	7F	FF	7F	FF	FF	FF	FF	FF	FF	FF	7F	FF	FF	FF
33	FF	FF	FF	FF	FF	FF	FF	7F	7F	7F	FF	FF	FF	FF	FF	FF	7F	7F	FF	FF	FF

Syntax PIXBLT L,L

Execution Source pixel array → Destination pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Operands L specifies that the source and destination pixel array starting addresses are given in linear format.

Description PIXBLT transfers and processes a source pixel array with a destination pixel array. This instruction operates on two-dimensional arrays of pixels using linear starting addresses for both the source and the destination. As the PixBlt proceeds, the source pixels are combined with the corresponding destination pixels based on the selected graphics operations.

Note that the instruction is entered as `PIXBLT L,L`. The following set of implied operands govern the operation of the instruction and define the source and destination arrays.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B0†‡	SADDR	Linear	Source pixel array starting address
B1†	SPTCH	Linear	Source pixel array pitch
B2†‡	DADDR	Linear	Destination pixel array starting address
B3	DPTCH	Linear	Destination pixel array pitch
B7	DYDX	XY	Pixel array dimensions (rows:columns)
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP– Pixel processing operations (22 options) T – Transparency operation PBH– Bit BLT horizontal direction PBV– Bit BLT vertical direction
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† These registers are changed by PIXBLT execution.
‡ You must adjust SADDR and DADDR to correspond to the corner selected by the values of PBH and PBV. See **Corner Adjust** below for additional information.

Source Array The source pixel array for the processing operation is defined by the contents of the SADDR, SPTCH, and DYDX registers:

- At the outset of the instruction, SADDR contains the **linear** address of the pixel at the appropriate starting corner of the array as determined by the PBH and PBV bits in the CONTROL I/O register. (See **Corner Adjust** below.)
- SPTCH contains the linear difference in the starting addresses of adjacent rows of the source array. SPTCH must be a multiple of 16.

- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of pixels per row.

During instruction execution, SADDR points to the next pixel (or word of pixels) to be read from the source array. When the block transfer is complete, SADDR points to the starting address of the next set of 32 pixels that would have been moved had the block transfer continued.

Destination Array

The location of the destination pixel array is defined by the contents of the DADDR, DPTCH, and DYDX registers:

- At the outset of the instruction, DADDR contains the **linear** address of the pixel at the appropriate starting corner of the array as determined by the PBH and PBV bits in the CONTROL I/O register. (See **Corner Adjust** below.)
- DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array. DPTCH must be a multiple of 16.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns.

During instruction execution, DADDR points to the next pixel (or word of pixels) to be modified in the destination array. When the block transfer is complete, DADDR points to the linear address of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Corner Adjust

The PBH and PBV bits in the CONTROL I/O register govern the direction of the PixBlt. If the source and destination arrays overlap, then PBH and PBV should be set to prevent any portion of the source array from being overwritten before it is moved.

However, this instruction is unique because the corner adjust is not automatic; the starting corners of both the source and destination arrays must be explicitly set to the alternate corner before instruction execution. Only the *direction* of the move is affected by the values of the PBH and PBV bits. This facility allows you to use corner adjust for screen definitions that do not lend themselves to XY addressing (those not binary powers of two). In effect, you supply your own corner adjust operation in software and the PixBlt instruction provides directional control. To use this feature, you must set both SADDR and DADDR to correspond to the corner selected by PBH and PBV.

- For **PBH** = **0** and **PBV** = **0**, SADDR and DADDR should be set as normally for linear PixBlts. Both registers should be set to correspond to the linear address of the **first** pixel on the **first** line of the array (that is, the pixel with the lowest address).

- For **PBH** = **0** and **PBV** = **1**, SADDR and DADDR should be set to correspond to the linear address of the **first** pixel on the **last** line of the array. In other words,

SADDR = (linear address of 1st pixel in source array) + (DY × SPTCH)

and

DADDR = (linear address of 1st pixel in dest. array) + (DY × DPTCH)

- For **PBH** = **1** and **PBV** = **0**, SADDR and DADDR should be set to correspond to the linear address of the *pixel following* the **last** pixel on the **first** line of the array. In other words,

SADDR = (linear address of 1st pixel in source array) + (DX × PSIZE)

and

DADDR = (linear address of 1st pixel in dest. array) + (DX × PSIZE)

- For **PBH** = **1** and **PBV** = **1**, SADDR and DADDR should be set to correspond to the linear address of the *pixel following* the **last** pixel on the **last** line of the array. In other words,

SADDR = (linear address of 1st pixel in source array) + (DY × SPTCH) + (DX × PSIZE)

and

DADDR = (linear address of 1st pixel in dest. array) + (DY × DPTCH) + (DX × PSIZE)

Window Checking

Window operations are not enabled for this instruction. The contents of the WSTART and WEND registers are ignored.

Pixel Processing

Pixel processing can be used with this instruction. The PPOP field of the CONTROL I/O register specifies the pixel processing operation that will be applied to pixels as they are processed with the destination array. There are 16 Boolean and 6 arithmetic operations; the default case at reset is the *replace* (S → D) operation. Note that the data is read through the plane mask and then processed. The 6 arithmetic operations do not operate with pixel sizes of 1 or 2 bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Transparency

Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it expands and processes the source data. At reset, the default case for transparency is *off*.

Plane Mask

The plane mask is enabled for this instruction.

Interrupts

This instruction can be interrupted at a word or row boundary of the destination array. When the PixBlt is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values. DADDR points to the linear address of the next word of pixels to be modified after the interrupt is processed. SADDR points to the address of the

next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the PixBlt correctly. You can inhibit the TMS34010 from resuming the PixBlt by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Shift Register Transfers

If the SRT bit in the DPYCTL I/O register is set, each memory read or write initiated by the PixBlt generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory clears or transfers. (Not all VRAMs support this capability.)

Words 1

Machine States

See Section 13.4, PIXBLT Instructions Timing.

Status Bits

N Undefined
C Undefined
Z Undefined
V Undefined

Examples

Before the PIXBLT instruction can be executed, the implied operand registers must be loaded with appropriate values. These PIXBLT examples use the following implied operand setup.

Register File B:
SADDR (B0) = >0000 2004
SPTCH (B1) = >0000 0080
DADDR (B2) = >0000 2228
DPTCH (B3) = >0000 0080
OFFSET (B4) = >0000 0000
DYDX (B7) = >0002 000D

I/O Registers:
PSIZE = >0004

Additional implied operand values are listed with each example.

For this example, assume that memory contains the following data before instruction execution.

Linear Address	Data							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>FFxx,	>FFFF,	>FFFF,	>xFFF,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>FFxx,	>FFFF,	>FFFF,	>xFFF,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 1

This example uses the *replace (S → D)* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>00xx,	>1110,	>2221,	>x332,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>00xx,	>1110,	>2221,	>x332,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 2

This example uses the *(D - S) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >4800 (T=0, W=00, PP=10010).

After instruction execution, memory will contain the following values:

Linear Address	Data							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>FFxx,	>EEEF,	>DDDE,	>xCCD,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>FFxx,	>EEEF,	>DDDE,	>xCCD,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 3

This example uses transparency. Before instruction execution, PMASK = > 0000 and CONTROL = > 0020 (T=1, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>FFxx,	>111F,	>2221,	>x332,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>FFxx,	>111F,	>2221,	>x332,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 4 This example uses plane masking; the MSB of each pixel is masked. Before instruction execution, PMASK = >8888 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data						
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx, >xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx, >xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx, >xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx, >xxxx
>02200	>xxxx,	>xxxx,	>88xx,	>9998,	>AAA9,	>xBBA,	>xxxx, >xxxx
>02280	>xxxx,	>xxxx,	>88xx,	>9998,	>AAA9,	>xBBA,	>xxxx, >xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx, >xxxx

Syntax **PIXBLT** L,XY

Execution Source pixel array → Destination pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0

Operands

L specifies that the source pixel array starting address is given in linear format.

XY specifies that the destination pixel array starting address is given in XY format.

Description PIXBLT transfers and processes a source pixel array with a destination pixel array. This instruction operates on two-dimensional arrays of pixels using a linear starting addresses for the source array and an XY address for the destination array. As the PixBlt proceeds, the source pixels are combined with the corresponding destination pixels based on the selected graphics operations.

Note that the instruction is entered as `PIXBLT L,XY`. The following set of implied operands govern the operation of the instruction and define the source and destination arrays.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B0†	SADDR	Linear	Source pixel array starting address
B1	SPTCH	Linear	Source pixel array pitch
B2†‡	DADDR	XY	Destination pixel array starting address
B3	DPTCH	Linear	Destination pixel array pitch
B4	OFFSET	Linear	Screen origin (0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner
B7‡	DYDX	XY	Pixel array dimensions (rows:columns)
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP– Pixel processing operations (22 options) W – Window operations T – Transparency operation PBH– PixBlt horizontal direction PBV– PixBlt vertical direction
>C0000130	CONVSP	XY-to-linear conversion (source pitch) Used for preclipping and corner adjust
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† These registers are changed by PIXBLT execution.
‡ Used for common rectangle function with window pick.

Source Array The source pixel array for the processing operation is defined by the contents of the SADDR, SPTCH, DYDX, and (potentially) CONVSP registers:

- At the outset of the instruction, SADDR contains the **linear** address of the pixel with the lowest address in the array.
- SPTCH contains the linear difference in the starting addresses of adjacent rows of the source array. SPTCH must be a multiple of 16. For window clipping or corner adjust, SPTCH must be a power of two and CONVSP must be set to correspond to the SPTCH value.
- CONVSP is computed by operating on the SPTCH register with the LMO instruction; it is used for the XY calculations involved in window clipping and corner adjust.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of pixels per row.

During instruction execution, SADDR points to the next pixel (or word of pixels) to be accessed in the source array. When the block transfer is complete, SADDR points to the linear address of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Destination Array The location of the destination pixel array is defined by the contents of the DADDR, DPTCH, CONVDP, OFFSET, and DYDX registers:

- At the outset of the instruction, DADDR contains the **XY** address of the pixel with the lowest address in the array. It is used with OFFSET and CONVDP to calculate the linear address of the starting location of the array.
- DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array (typically this is the screen pitch). DPTCH must be a power of two (greater than or equal to 16) and
- CONVDP must be set to correspond to the DPTCH value. CONVDP is computed by operating on the DPTCH register with the LMO instruction; it is used for the XY calculations involved in XY addressing, window clipping and corner adjust.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns.

During instruction execution, DADDR points to the **linear address** of next pixel (or word of pixels) to be accessed in the destination array. When the block transfer is complete, DADDR points to the **linear address** of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Corner Adjust The PBH and PBV bits in the CONTROL I/O register govern the direction of the PixBlt. If the source and destination arrays overlap, then PBH and PBV should be set to prevent any portion of the source array from being

overwritten before it is moved. This PixBlt performs the corner adjust function automatically under the control of the PBH and PBV bits. If PBV=1, SPTCH must be a power of two and CONVSP should be valid. The SADDR and DADDR registers should be set to correspond to the appropriate format address of the **first** pixel on the **first** line of the source (linear) and destination (XY) arrays, respectively.

Window Checking

Window checking can be used with this instruction by setting the two W bits in the CONTROL register to the desired value. If window checking mode 1, 2, or 3 is selected, the WSTART and WEND registers define the XY starting and ending corners of a rectangular window.

0 *No windowing.* The entire pixel array is drawn and the WVP and V bits are unaffected.

1 *Window hit.* No pixels are drawn. The V bit is set to 0 if any portion of the destination array lies within the window. Otherwise, the V bit is set to 1.

If the V bit is set to 0, the DADDR and DYDX registers are modified to correspond to the common rectangle formed by the intersection of the destination array with the rectangular window. DADDR is set to the XY address of the pixel in the starting corner of the common rectangle. DYDX is set to the X and Y dimensions of the common rectangle.

If the V bit is set to 1, the array lies entirely outside the window, and the values of DADDR and DYDX are indeterminate.

2 *Window miss.* If the array lies **entirely** within the active window, it is drawn and the V bit is set to 0. Otherwise, no pixels are drawn, the V and WVP bits are set to 1, and the instruction is aborted.

3 *Window clip.* The source and destination arrays are preclipped to the window dimensions. Only those pixels that lie within the common rectangle (corresponding to the intersection of the specified array and the window) are drawn. If any preclipping is required, the V bit is set to 1.

Pixel Processing

Pixel processing can be used with this instruction. The PPOP field of the CONTROL I/O register specifies the pixel processing operation that will be applied to pixels as they are processed with the destination array. There are 16 Boolean and 6 arithmetic operations; the default case at reset is the *replace* (S → D) operation. Note that the data is read through the plane mask and then processed. The 6 arithmetic operations do not operate with pixel sizes of 1 or 2 bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it expands and processes the source data. At reset, the default case for transparency is *off.*

Plane Mask The plane mask is enabled for this instruction.

Interrupts This instruction can be interrupted at a word or row boundary of the destination array. When the PixBlt is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values.

DADDR points to the linear address of the next word of pixels to be modified after the insterrupt is processed. SADDR points to the address of the next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the PixBlt correctly. You can inhibit the TMS34010 from resuming the PixBlt by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Shift Register Transfers

If the SRT bit in the DPYCTL I/O register is set, each memory read or write initiated by the PixBlt generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory clears or transfers. (Not all VRAMs support this capability.)

Words

1

Machine States

See PIXBLT Instructions Timing, Section 13.4.

Status Bits

N Undefined
C Undefined
Z Undefined
V If window clipping is enabled – 1 if a window violation occurs, *0* otherwise. Undefined if window clipping not enabled (W=00).

Examples

Before the PIXBLT instruction can be executed, the implied operand registers must be loaded with appropriate values. These PIXBLT examples use the following implied operand setup.

Register File B:		**I/O Registers:**	
SADDR (B0)	= >0000 2004	CONVDP	= >0017
SPTCH (B1)	= >0000 0080	PSIZE	= >0004
DADDR (B2)	= >0052 0007	PMASK	= >0000
DPTCH (B3)	= >0000 0100	CONTROL	= >0000
OFFSET (B4)	= >0001 0000		(W=00, T=0, PP=00000)
WSTART (B5)	= >0030 000C		
WEND (B6)	= >0053 0014		
DYDX (B7)	= >0003 0016		

Additional implied operand values are listed with each example.

For this example, assume that memory contains the following data before instruction execution.

Linear Address	**Data**
>02000	>3210, >7654, >BA98,>FEDC,>3210, >7654, >BA98,>FEDC
>02080	>3210, >7654, >BA98,>FEDC,>3210, >7654, >BA98,>FEDC
>02100	>3210, >7654, >BA98,>FEDC,>3210, >7654, >BA98,>FEDC
>15200 to >15480	>8888

Example 1

This example uses the *replace (S → D)* pixel processing operation. Before instruction execution, PMASK = >7777 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data
>15200	>8888, >1888, >5432, >9876, >DCBA,>10FE, >5432, >8886
>15300	>8888, >1888, >5432, >9876, >DCBA,>10FE, >5432, >8886
>15400	>8888, >1888, >5432, >9876, >DCBA,>10FE, >5432, >8886

XY Addressing

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	8	8	8	8	8	8	8	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	8	8	8	8
53	8	8	8	8	8	8	8	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	8	8	8	8
54	8	8	8	8	8	8	8	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	8	8	8	8

Example 2

This example uses the *(D subs S) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >4C00 (T=0, W=00, PP=10011).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	8	8	8	8	8	8	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	8	7	6	5	4	3	2	8	8	8
53	8	8	8	8	8	8	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	8	7	6	5	4	3	2	8	8	8
54	8	8	8	8	8	8	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	8	7	6	5	4	3	2	8	8	8

Example 3

This example uses transparency with the *(D subs S) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >4C20 (T=1, W=00, PP=10011).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	8	8	8	8	8	8	8	7	6	5	4	3	2	1	8	8	8	8	8	8	8	8	8	7	6	5	4	3	2	8	8	8
53	8	8	8	8	8	8	8	7	6	5	4	3	2	1	8	8	8	8	8	8	8	8	8	7	6	5	4	3	2	8	8	8
54	8	8	8	8	8	8	8	7	6	5	4	3	2	1	8	8	8	8	8	8	8	8	8	7	6	5	4	3	2	8	8	8

Example 4 This example uses window operation 3 (the destination is clipped). Before instruction execution, PMASK = >0000 and CONTROL = >00C0 (T=0, W=11, PP=00000).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	8	8	8	8	8	8	8	8	8	8	8	8	6	7	8	9	A	B	C	D	E	8	8	8	8	8	8	8	8	8	8	8
53	8	8	8	8	8	8	8	8	8	8	8	8	6	7	8	9	A	B	C	D	E	8	8	8	8	8	8	8	8	8	8	8
54	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8

Example 5 This example uses plane masking; the most significant bit is masked. Before instruction execution, PMASK = >8888 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
52	8	8	8	8	8	8	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	8	8	8
53	8	8	8	8	8	8	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	8	8	8
54	8	8	8	8	8	8	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	8	8	8

Syntax **PIXBLT** XY,L

Execution Source pixel array → Destination pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0

Operands

XY specifies that the source pixel array starting address is given in XY format.

L specifies that the destination pixel array starting address is given in linear format.

Description

PIXBLT transfers and processes a source pixel array with a destination pixel array. This instruction operates on two-dimensional arrays of pixels using an XY starting address for the source pixel array and a linear address for the destination array. As the PixBlt proceeds, the source pixels are combined with the corresponding destination pixels based on the selected graphics operations.

Note that the instruction is entered as `PIXBLT XY,L`. The following set of implied operands govern the operation of the instruction and define the source and destination arrays.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B0†	SADDR	XY	Source pixel array starting address
B1	SPTCH	Linear	Source pixel array pitch
B2†	DADDR	Linear	Destination pixel array starting address
B3	DPTCH	Linear	Destination pixel array pitch
B4	OFFSET	Linear	Screen origin (0,0)
B7	DYDX	XY	Pixel array dimensions (rows:columns)
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP – Pixel processing operations (22 options) T – Transparency operation PBH – PixBlt horizontal direction PBV – PixBlt vertical direction
>C0000130	CONVSP	XY-to-linear conversion (source pitch) Used for XY operations
>C0000140	CONVDP	XY-to-linear conversion (destination pitch) Used for XY operations
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† These registers are changed by PIXBLT execution.

Source Array The source pixel array for the processing operation is defined by the contents of the SADDR, SPTCH, CONVSP, OFFSET, and DYDX registers:

- At the outset of the instruction, SADDR contains the **XY** address of the pixel with the lowest address in the array. It is used with OFFSET and CONVSP to calculate the linear address of the starting location of the array.
- SPTCH contains the linear difference in the starting addresses of adjacent rows of the source array (typically this is the screen pitch). SPTCH must be a power of two (greater than or equal to 16) and
- CONVSP must be set to correspond to the SPTCH value. CONVSP is computed by operating on the SPTCH register with the LMO instruction; it is used for the XY calculations involved in XY addressing, window clipping and corner adjust.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns.

During instruction execution, SADDR points to the next pixel (or word of pixels) to be accessed from the source array. When the block transfer is complete, SADDR points to the **linear address** of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Destination Array The location of the destination pixel array is defined by the contents of the DADDR, DPTCH, DYDX, and (potentially) CONVDP registers:

- At the outset of the instruction, DADDR contains the **linear** address of the pixel with the lowest address in the array.
- DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array. DPTCH must be a multiple of 16. For window clipping or corner adjust, DPTCH must be a power of two and CONVDP must be set to correspond to the DPTCH value.
- CONVDP is computed by operating on the DPTCH register with the LMO instruction; it is used for the XY calculations involved in window clipping and corner adjust.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns.

During instruction execution, DADDR points to the next pixel (or word of pixels) to be modified in the destination array. When the block transfer is complete, DADDR points to the linear address of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Corner Adjust The PBH and PBV bits in the CONTROL I/O register *govern* the direction of the PixBlt. If the source and destination arrays overlap, then PBH and PBV should be set to prevent any portion of the source array from being overwritten before it is moved. This PixBlt performs the corner adjust function automatically under the control of the PBH and PBV bits. If PBV=1, DPTCH must be a power of two and CONVDP must be valid. The SADDR and DADDR registers should be set to correspond to the appropriate format address of the **first** pixel on the **first** line of the source (XY) and destination (linear) arrays, respectively.

Window Checking Window operations are not enabled for this instruction. The contents of the WSTART and WEND registers are ignored.

Pixel Processing Pixel processing can be used with this instruction. The PPOP field of the CONTROL I/O register specifies the pixel processing operation that will be applied to pixels as they are processed with the destination array. There are 16 Boolean and 6 arithmetic operations; the default case at reset is the S → D operation. Note that the data is read through the plane mask and then processed. The 6 arithmetic operations do not operate with pixel sizes of one or two bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it expands and processes the source data. At reset, the default case for transparency is *off*.

Plane Mask The plane mask is enabled for this instruction.

Interrupts This instruction can be interrupted at a word or row boundary of the destination array. When the PixBlt is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values. DADDR points to the linear address of the next word of pixels to be modified after the insterrupt is processed. SADDR points to the address of the next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the PixBlt correctly. You can inhibit the TMS34010 from resuming the PixBlt by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Shift Register Transfers If the SRT bit in the DPYCTL I/O register is set, each memory read or write initiated by the PixBlt generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory *clears* or transfers. (Not all VRAMs support this capability.)

Words 1

Machine States See PIXBLT Instructions Timing, Section 13.4.

Status Bits

- **N** Undefined
- **C** Undefined
- **Z** Undefined
- **V** Undefined

Examples

Before the PIXBLT instruction can be executed, the implied operand registers must be loaded with appropriate values. These PIXBLT examples use the following implied operand setup.

Register File B:		**I/O Registers:**	
SADDR (B0)	= >00400001	CONVSP	= >0018
SPTCH (B1)	= >00000080	PSIZE	= >004
DADDR (B2)	= >00002228		
DPTCH (B3)	= >00000080		
OFFSET (B4)	= >00000000		
DYDX (B7)	= >0002000D		

Additional implied operand values are listed with each example.

For this example, assume that memory contains the following data before instruction execution.

Linear Address	**Data**							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>FFxx,	>FFFF,	>FFFF,	>xFFF,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>FFxx,	>FFFF,	>FFFF,	>xFFF,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 1

This example uses the *replace (S → D)* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	**Data**							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>00xx,	>1110,	>2221,	>x332,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>00xx,	>1110,	>2221,	>x332,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 2 This example uses the *0s* → *D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >0C00 (T=0, W=00, PP=00011).

After instruction execution, memory will contain the following values:

Linear Address	Data							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>00xx,	>0000,	>0000,	>x000,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>00xx,	>0000,	>0000,	>x000,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 3 This example uses transparency. Befrore instruction execution, PMASK = > 0000 and CONTROL = > 0020 (T=1, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>FFxx,	>111F,	>2221,	>x332,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>FFxx,	>111F,	>2221,	>x332,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Example 4 This example uses plane masking; the two MSBs of each pixel are masked. Before instruction execution, PMASK = >CCCC and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data							
>02000	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02080	>000x,	>1111,	>2222,	>xx33,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02100	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02180	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx
>02200	>xxxx,	>xxxx,	>CCxx,	>DDDC,	>EEED,	>xFFE,	>xxxx,	>xxxx
>02280	>xxxx,	>xxxx,	>CCxx,	>DDDC,	>EEED,	>xFFE,	>xxxx,	>xxxx
>02300	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx,	>xxxx

Syntax **PIXBLT XY,XY**

Execution Source pixel array → Destination pixel array (with processing)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0

Operands **XY** specifies that the source and destination pixel array starting addresses are given in XY format.

Description PIXBLT transfers and processes a source pixel array with a destination pixel array. This instruction operates on two-dimensional arrays of pixels using XY starting addresses for both the source and destination pixel arrays. As the PixBlt proceeds, the source pixels are combined with the corresponding destination pixels based on the selected graphics operations.

Note that the instruction is entered as `PIXBLT XY,XY.` the destination. The following set of implied operands govern the operation of the instruction and define the source and destination arrays.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B0†	SADDR	XY	Source pixel array starting address
B1	SPTCH	Linear	Source pixel array pitch
B2†‡	DADDR	XY	Destination pixel array starting address
B3	DPTCH	Linear	Destination pixel array pitch
B4	OFFSET	Linear	Screen origin (0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner
B7‡	DYDX	XY	Pixel array dimensions (rows:columns)
B10–B14†			Reserved registers

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP – Pixel processing operations (22 options) W – Window clipping or pick operation T – Transparency operation PBH – PixBlt horizontal direction PBV – PixBlt vertical direction
>C0000130	CONVSP	XY-to-linear conversion (source pitch)
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

† These registers are changed by PIXBLT execution.
‡ Used for common rectangle function with window pick.

Source Array The source pixel array for the processing operation is defined by the contents of the SADDR, SPTCH, CONVSP, OFFSET, and DYDX registers:

- At the outset of the instruction, SADDR contains the **XY** address of the pixel with the lowest address in the array. It is used with OFFSET and CONVSP to calculate the linear address of the starting location of the array.
- SPTCH contains the linear difference in the starting addresses of adjacent rows of the source array (typically this is the screen pitch). SPTCH must be a power of two (greater than or equal to 16) and CONVSP must be set to correspond to the SPTCH value.
- CONVSP is computed by operating on the SPTCH register with the LMO instruction; it is used for the XY calculations involved in XY addressing, window clipping and corner adjust.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns.

During instruction execution, SADDR points to the next pixel (or word of pixels) to be read from the source array. When the block transfer is complete, SADDR points to the **linear address** of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Destination Array The location of the destination pixel array is defined by the contents of the DADDR, DPTCH, CONVDP, OFFSET, and DYDX registers:

- At the outset of the instruction, DADDR contains the **XY** address of the pixel with the lowest address in the array. It is used with OFFSET and CONVDP to calculate the linear address of the starting location of the array.
- DPTCH contains the linear difference in the starting addresses of adjacent rows of the destination array (typically this is the screen pitch). DPTCH must be a power of two (greater than or equal to 16) and CONVDP must be set to correspond to the DPTCH value.
- CONVDP is computed by operating on the DPTCH register with the LMO instruction; it is used for the XY calculations involved in XY addressing, window clipping and corner adjust.
- DYDX specifies the dimensions of both the source and destination arrays in pixels. The DY portion of DYDX contains the number of rows in the array, while the DX portion contains the number of columns.

During instruction execution, DADDR points to the next pixel (or word of pixels) to be read from the destination array. When the block transfer is complete, DADDR points to the **linear address** of the first pixel on the **next** row of pixels that would have been moved had the block transfer continued.

Window Checking

Window checking can be used with this instruction by setting the two W bits in the CONTROL register to the desired value. If window checking mode 1, 2, or 3 is selected, the WSTART and WEND registers define the XY starting and ending corners of a rectangular window.

0 *No windowing.* The entire pixel array is drawn and the WVP and V bits are unaffected.

1 *Window hit.* No pixels are drawn. The V bit is set to 0 if any portion of the destination array lies within the window. Otherwise, the V bit is set to 1.

If the V bit is set to 0, the DADDR and DYDX registers are modified to correspond to the common rectangle formed by the intersection of the destination array with the rectangular window. DADDR is set to the XY address of the pixel in the starting corner of the common rectangle. DYDX is set to the X and Y dimensions of the common rectangle.

If the V bit is set to 1, the array lies entirely outside the window, and the values of DADDR and DYDX are indeterminate.

2 *Window miss.* If the array lies **entirely** within the active window, it is drawn and the V bit is set to 0. Otherwise, no pixels are drawn, the V and WVP bits are set to 1, and the instruction is aborted.

3 *Window clip.* The source and destination arrays are preclipped to the window dimensions. Only those pixels that lie within the common rectangle (corresponding to the intersection of the specified array and the window) are drawn. If any preclipping is required, the V bit is set to 1.

Pixel Processing

Pixel processing can be used with this instruction. The PPOP field of the CONTROL I/O register specifies the pixel processing operation that will be applied to pixels as they are processed with the destination array. There are 16 Boolean and 6 arithmetic operations; the default case at reset is the *replace* (S → D) operation. Note that the data is read through the plane mask and then processed. The 6 arithmetic operations do not operate with pixel sizes of one or two bits per pixel. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Corner Adjust

The PBH and PBV bits in the CONTROL I/O register govern the direction of the PixBlt. If the source and destination arrays overlap, then PBH and PBV should be set to prevent any portion of the source array from being overwritten before it is moved. This PixBlt performs the corner adjust function automatically under the control of the PBH and PBV bits. The SADDR and DADDR registers should be set to correspond to the appropriate format address of the **first** pixel on the **first** line of the source (XY) and destination (XY) arrays, respectively.

Transparency

Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it expands and processes the source data. At reset, the default case for transparency is *off*.

Plane Mask

The plane mask is enabled for this instruction.

Interrupts

This instruction can be interrupted at a word or row boundary of the destination array. When the PixBlt is interrupted, the TMS34010 sets the PBX bit in the status register and then pushes the status register on the stack. At this time, DPTCH, SPTCH, and B10–B14 contain intermediate values. DADDR points to the linear address of the next word of pixels to be modified after the insterrupt is processed. SADDR points to the address of the next 32 pixels to be read from the source array after the interrupt is processed.

Before executing the RETI instruction to return from the interrupt, restore any B-file registers that were modified (also restore the CONTROL register if it was modified). This allows the TMS34010 to resume the PixBlt correctly. You can inhibit the TMS34010 from resuming the PixBlt by executing an `RETS 2` instruction instead of RETI; however, SPTCH, DPTCH, and B10–B14 will contain indeterminate values.

Shift Register Transfers

If the SRT bit in the DPYCTL I/O register is set, each memory read or write initiated by the PixBlt generates a shift register transfer read or write cycle at the selected address. This operation can be used for bulk memory clears or transfers. (Not all VRAMs support this capability.)

Words

1

Machine States

See Section 13.4, PIXBLT Instructions Timing.

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V If window clipping is enabled – 1 if a window violation occurs, *0* otherwise. Unaffected if window clipping not enabled.

Examples

Before the PIXBLT instruction can be executed, the implied operand registers must be loaded with appropriate values. These PIXBLT examples use the following implied operand setup.

Register File B:		**I/O Registers:**	
SADDR (B0)	= >0020 0004	CONVSP	= >0016
SPTCH (B1)	= >0000 0200	CONVDP	= >0016
DADDR (B2)	= >0041 0004	PSIZE	= >0004
DPTCH (B3)	= >0000 0200	PMASK	= >0000
OFFSET(B4)	= >0001 0000	CONTROL	= >0000
WSTART(B5)	= >0030 0009		(W=00, T=0, PP=00000)
WEND (B6)	= >0042 0012		
DYDX (B7)	= >0003 0016		

Additional implied operand values are listed with each example. For this example, assume that memory contains the following data before instruction execution.

Linear Address	**Data**
>14000	>3210, >7654, >BA98, >FEDC, >3210, >7654, >BA98, >FEDC
>14200	>3210, >7654, >BA98, >FEDC, >3210, >7654, >BA98, >FEDC
>14400	>3210, >7654, >BA98, >FEDC, >3210, >7654, >BA98, >FEDC
>18200 to >18680	>3333

Example 1

This example uses the *replace (S →D)* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

Linear Address	Data
>18200	>3333, >7654, >BA98,>FEDC,>3210, >7654, >3398, >3333
>18400	>3333, >7654, >BA98,>FEDC,>3210, >7654, >3398, >3333
>18600	>3333, >7654, >BA98,>FEDC,>3210, >7654, >3398, >3333

XY Addressing

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
41	3	3	3	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	3	3	3	3	3	3
42	3	3	3	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	3	3	3	3	3	3
43	3	3	3	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	3	3	3	3	3	3

Example 2

This example uses the *(D adds S) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >4400 (T=0, W=00, PP=10001).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
41	3	3	3	3	7	8	9	A	B	C	D	E	F	F	F	F	3	4	5	6	7	8	9	A	B	C	3	3	3	3	3	3
42	3	3	3	3	7	8	9	A	B	C	D	E	F	F	F	F	3	4	5	6	7	8	9	A	B	C	3	3	3	3	3	3
43	3	3	3	3	7	8	9	A	B	C	D	E	F	F	F	F	3	4	5	6	7	8	9	A	B	C	3	3	3	3	3	3

Example 3

This example uses transparency and the *(D SUBS S) → D* pixel processing operation. Before instruction execution, PMASK = >0000 and CONTROL = >4C20 (T=1, W=00, PP=10011).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
41	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3
42	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3
43	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3

Example 4

This example uses window operation 3 (the destination is clipped). Before instruction execution, PMASK = >0000 and CONTROL = >00C0 (T=0, W=11, PP=00000).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
41	3	3	3	3	3	3	3	3	3	9	A	B	C	D	E	F	0	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3
42	3	3	3	3	3	3	3	3	3	9	A	B	C	D	E	F	0	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3
43	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Example 5

This example uses plane masking; the third least significant bit is masked. Before instruction execution, PMASK = >5555 and CONTROL = >0000 (T=0, W=00, PP=00000).

After instruction execution, memory will contain the following values:

X Address

Y Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
41	3	3	3	3	1	1	3	3	9	9	B	B	9	9	B	B	1	1	3	3	1	1	3	3	9	9	3	3	3	3	3	3
42	3	3	3	3	1	1	3	3	9	9	B	B	9	9	B	B	1	1	3	3	1	1	3	3	9	9	3	3	3	3	3	3
43	3	3	3	3	1	1	3	3	9	9	B	B	9	9	B	B	1	1	3	3	1	1	3	3	9	9	3	3	3	3	3	3

Syntax **PIXT** *<Rs>*,**<Rd>*

Execution (pixel)Rs → (pixel)*Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	Rs				R	Rd			

Operands

Rs The source pixel is right justified in the specified register.

***Rd** *Destination register indirect*. The destination location is at the **linear** memory address contained in the specified register.

Description PIXT transfers a pixel from the source register to the **linear** memory address contained in the destination register. The source pixel is the 1, 2, 4, 8, or 16 LSBs of the source register, depending on the pixel size specified in the PSIZE I/O register. The source and destination registers must be in the same register file.

Implied Operands

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP– Pixel processing operations (22 options) T –Transparency operation
>C0000150	PSIZE	Pixel size (1,2,4,6,8,16)
>C0000160	PMASK	Plane mask – pixel format

Pixel Processing The PP field of the CONTROL I/O register selects the pixel processing operation to be applied to the pixel as it is transferred to the destination location. The default case at reset is the pixel processing *replace* operation. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Window Checking Window checking **cannot** be used with this instruction. The W bits are ignored.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it processes the source data. At reset, the default case for transparency is *off*.

Plane Mask The plane mask is enabled for this instruction.

Words 1

Machine States

Pixel Processing Operation							
PSIZE	**Replace**	**Boolean**	**ADD**	**ADDS**	**SUB**	**SUBS**	**MIN/MAX**
1,2,4,8	2+(3),8	4+(3),10	4+(3),11	5+(3),11	5+(3),12	6+(3),11	5+(3),10
16	2+(1),6	4+(1),8	4+(1),8	5+(1),9	5+(1),9	6+(1),10	5+(1),9

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples `PIXT A0,*A1`

Before							**After**		
	A0	**A1**	**@>20500**	**PSIZE**	**PP**	**T**	**PMASK**	**@>20500**
1)	>0000 FFFF	>0002 0500	>0000	>0001	00000	0	>0000	>0001
1)	>0000 FFFF	>0002 0500	>0000	>0002	00000	0	>0000	>0003
1)	>0000 FFFF	>0002 0500	>0000	>0004	00000	0	>0000	>000F
1)	>0000 FFFF	>0002 0500	>0000	>0008	00000	0	>0000	>00FF
1)	>0000 FFFF	>0002 0500	>0000	>0010	00000	0	>0000	>FFFF
1)	>0000 0006	>0002 0508	>0000	>0004	00000	0	>0000	>0600
2)	>0000 0006	>0002 0508	>0300	>0004	01010	0	>0000	>0500
3)	>0000 0006	>0002 0508	>0100	>0004	00001	0	>0000	>0000
4)	>0000 0006	>0002 0508	>0100	>0004	00001	1	>0000	>0100
5)	>0000 0006	>0002 0508	>0000	>0004	00000	0	>AAAA	>0400

Notes:

1) S replaces D
2) (S XOR D) replaces D
3) (S AND D) = 0, transparency is off, D is replaced
4) (S + D) = 0, transparency is on, D is not replaced
5) S replaces unmasked bits of D

PIXT Pixel Transfer - *Register to Indirect XY*

Syntax **PIXT** *<Rs>*,**<Rd>*.**XY**

Execution (pixel)Rs → (pixel)*Rd.XY

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	Rs				R	Rd			

Operands

Rs The source pixel is right justified in the specified register.

***Rd.XY** *Destination register indirect in XY format.* The destination location is the XY address contained in the specified register. The X value occupies the 16 LSBs of the register and the Y value occupies the 16 MSBs.

Description PIXT transfers a pixel from the source register to the XY memory address contained in the destination register. The source pixel is the 1, 2, 4, 8, or 16 LSBs of the source register, depending on the pixel size specified in the PSIZE I/O register. The source and destination registers must be in the same register file.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B3	DPTCH	Linear	Destination pitch
B4	OFFSET	Linear	Screen origin (0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP – Pixel processing operations (22 options) W – Window clipping or pick operation T – Transparency operation
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

Window Checking Window checking can be selected by setting the W bits in the CONTROL register to the desired value. If one of the three active window modes (1, 2, or 3) is selected, the WSTART and WEND registers define the starting and ending window corners. When an attempt is made to write a pixel inside or outside a window, the results depend on the selected window checking mode:

0 *No window checking.* The pixel is drawn and the WVP and V bits are unaffected.

1 *Window hit.* No pixels are drawn. The V bit is set to 0 if the pixel lies within the window; otherwise, it is set to 1.

2 *Window miss.* If the pixel lies outside the window, the V and WVP bits are set to 1 and the instruction is aborted (no pixels are drawn). Otherwise, the pixel is drawn and the V bit is set to 0.

3 *Window clip.* If the pixel lies outside the window, the V bit is set to 1 and the instruction is aborted (no pixels are drawn). Otherwise, the pixel is drawn and the V bit is set to 0.

For more information, see Section 7.10, Window Checking, on page 7-25.

Pixel Processing

The PP field of the CONTROL I/O register specifies the pixel processing operation of that will be applied to the pixel as it is transferred to the destination location. The default case at reset is the pixel processing *replace* operation. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Transparency

Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it processes the source data. At reset, the default case for transparency is *off*.

Plane Mask

The plane mask is enabled for this instruction.

Words

1

***Machine* States**

Pixel Processing Operation								Window Violation		
PSIZE	Replace	Boolean	ADD	ADDS	SUB	SUBS	MIN/MAX	W=1	W=2	W=3
1,2,4,8	4+(3),10	6+(3),12	6+(3),12	7+(3),13	7+(3),13	8+(3),14	7+(3),13	6,9	6,9	4,7
16	4+(1),8	6+(1),10	6+(1),10	7+(1),11	7+(1),11	8+(1),12	7+(1),11	6,9	6,9	4,7

Status Bits

N Unaffected
C Unaffected
Z Unaffected
V *1* if window clipping enabled and window violation or pick occurs, *0* if no window violation occurs. Unaffected if window clipping is not enabled.

Examples Before the PIXT instruction can be executed, the implied operand registers must be loaded with appropriate values. These PIXT examples use the following implied operand setup.

Register File B:
DPTCH (B3) = >00000800
OFFSET (B4) = >00000000
WTART (B5) = >00300020
WEND (B6) = >00500142

I/O Registers:
CONVDP = >0014

```
PIXT  A0,*A1.XY
```

	Before								**After**
	A0	**A1**	**@>20500**	**PSIZE**	**PP**	**W**	**T**	**PMASK**	**@>20500**
1)	>0000 FFFF	>0040 0500	>0000	>0001	00000	00	0	>0000	>0001
1)	>0000 FFFF	>0040 0280	>0000	>0002	00000	00	0	>0000	>0003
1)	>0000 FFFF	>0040 0140	>0000	>0004	00000	00	0	>0000	>000F
1)	>0000 FFFF	>0040 00A0	>0000	>0008	00000	00	0	>0000	>00FF
1)	>0000 FFFF	>0040 0050	>0000	>0010	00000	00	0	>0000	>FFFF
1)	>0000 0006	>0040 0142	>0000	>0004	00000	00	0	>0000	>0600
2)	>0000 0006	>0040 0142	>0300	>0004	01010	00	0	>0000	>0500
3)	>0000 0006	>0040 0142	>0100	>0004	00001	00	0	>0000	>0000
4)	>0000 0006	>0040 0142	>0100	>0004	00001	00	1	>0000	>0100
5)	>0000 0006	>0040 0142	>0000	>0004	00000	00	0	>AAAA	>0400
6)	>0000 0006	>0040 0142	>0000	>0004	00000	11	0	>0000	>0600
7)	>0000 0006	>0040 0143	>0000	>0004	00000	11	0	>0000	>0000
8)	>0000 0006	>0040 0143	>0000	>0004	00000	10	0	>0000	>0000

XY Address in A1 = Linear Address >20500

Notes:

1) S replaces D
2) (S XOR D) replaces D
3) (S AND D) = 0, transparency is off, D is replaced
4) (S + D) = 0, transparency is on, D not replaced
5) S replaces unmasked bits of D
6) Window Option = 3, D inside window, S replaces D
7) Window Option = 3, D outside window, D not replaced, V bit set in status register
8) Window Option = 2, D outside window, D not replaced, WV interrupt generated, V bit set in status register

Syntax **PIXT** *<*Rs*>*,*<*Rd*>*

Execution (pixel)*Rs → (pixel)Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	Rs				R	Rd			

Operands ***Rs** *Source register indirect.* The source pixel is located at the **linear** memory address contained in the specified register.

Description PIXT transfers a pixel from the **linear** memory address contained in the source register to the destination register. When the pixel is moved into the register, it is right justified and zero extended to 32 bits according to the pixel size specified in the PSIZE I/O register. The source and destination registers must be in the same register file.

Implied Operands

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C0000150	PSIZE	Pixel size (1,2,4,6,8,16)
>C0000160	PMASK	Plane mask – pixel format

Window Checking Window checking **cannot** be used with this instruction. The W bits are ignored.

Pixel Processing Pixel processing **cannot** be used with this instruction.

Transparency Transparency **cannot** be used with this instruction.

Plane Mask The plane mask is enabled for this instruction.

Words 1

Machine States 4,7

Status Bits
N Undefined
C Undefined
Z Undefined
V Set to 1 if the pixel is 1, set to 0 if the pixel is 0.

Examples Assume that memory contains the following values:

Address	Data
@>20500	>FFFF
@>20510	>3333

```
PIXT  *A0,A1
```

Before			After
A0	**PSIZE**	**PMASK**	**A1**
>0002 0500	>0001	>0000	>0000 0001
>0002 0500	>0001	>FFFF	>0000 0000
>0002 0500	>0002	>0000	>0000 0003
>0002 0500	>0002	>5555	>0000 0002
>0002 0500	>0004	>0000	>0000 000F
>0002 0510	>0004	>9999	>0000 0002
>0002 0500	>0008	>0000	>0000 00FF
>0002 0510	>0008	>5454	>0000 0023
>0002 0500	>0010	>0000	>0000 FFFF
>0002 0500	>0010	>BA98	>0000 4567
>0002 0510	>0010	>BA98	>0000 0123

Syntax PIXT *<*Rs*>,*<*Rd*>

Execution pixel(*Rs) → pixel(*Rd)

Encoding

15	14	13	12	11	10	9	8–5	4	3–0
1	1	1	1	1	1	0	Rs	R	Rd

Operands

***Rs** *Source register indirect.* The source pixel is located at the **linear** memory address contained in the specified register.

***Rd** *Destination register indirect.* The destination location is at the **linear** memory address contained in the specified register.

Description PIXT transfers a pixel from the **linear** memory address contained in the source register to the **linear** memory address contained in the destination register. The source and destination registers must be in the same register file.

Implied Operands

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP– Pixel processing operations (22 options) T – Transparency operation
>C0000150	PSIZE	Pixel size (1,2,4,6,8,16)
>C0000160	PMASK	Plane mask – pixel format

Pixel Processing The PP field of the CONTROL I/O register selects the pixel processing operation that will be applied to the pixels as they are transferred to the destination array. The default case at reset is the pixel processing *replace* operation. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Window Checking Window checking **cannot** be used with this instruction. The W bits are ignored.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it processes the source data. At reset, the default case for transparency is *off.*

Plane Mask The plane mask is enabled for this instruction.

Words 1

Machine States

	Pixel Processing Operation							Window Violation		
PSIZE	**Replace**	**Boolean**	**ADD**	**ADDS**	**SUB**	**SUBS**	**MIN/MAX**	**W=1**	**W=2**	**W=3**
1,2,4,8	4+(3),10	6+(3),12	6+(3),12	7+(3),13	7+(3),13	8+(3),14	7+(3),13	–	–	–
16	4+(1),8	6+(1),10	6+(1),10	7+(1),11	7+(1),11	8+(1),12	7+(1),11	–	–	–

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** Unaffected

Examples `PIXT *A0,*A1`

	Before							After	
	A0	**A1**	**@>20500**	**PSIZE**	**PP**	**T**	**PMASK**	**@>20500**	**@>20510**
1)	>0002 0500	>0002 0508	>000F	>0001	00000	0	>0000	>010F	xxxx
1)	>0002 0500	>0002 0508	>000F	>0002	00000	0	>0000	>030F	xxxx
1)	>0002 0500	>0002 0508	>000F	>0004	00000	0	>0000	>0F0F	xxxx
1)	>0002 0500	>0002 0508	>00EF	>0008	00000	0	>0000	>EFEF	xxxx
1)	>0002 0500	>0002 0508	>1234	>0010	00000	0	>0000	>3434	>xx12
2)	>0002 0500	>0002 0508	>030F	>0004	01010	0	>0000	>0C0F	xxxx
3)	>0002 0500	>0002 0508	>010E	>0004	00001	0	>0000	>000E	xxxx
4)	>0002 0500	>0002 0508	>020E	>0004	00001	1	>0000	>020E	xxxx
5)	>0002 0500	>0002 0508	>000F	>0004	00000	0	>AAAA	>050F	xxxx

Notes:

1) S replaces D
2) (S XOR D) replaces D
3) (S AND D) = 0, transparency is off, D is replaced
4) (S + D) = 0, transparency is on, D not replaced
5) S replaces unmasked bits of D

Syntax **PIXT** *<*Rs*>*.XY,<*Rd*>

Execution (pixel)*Rs.XY → (pixel)Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	Rs				R	Rd			

Operands ***Rs.XY** *Source register indirect in XY format.* The source operand is at the XY memory address contained in the specified register. The X value occupies the 16 LSBs of the register and the Y value occupies the 16 MSBs.

Description PIXT transfers a pixel from the XY memory address contained in the source register to the destination register. When the pixel is moved into the register, it is right justified and zero extended to 32 bits according to the pixel size specified in the PSIZE I/O register. The source and destination registers must be in the same register file.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B3	DPTCH	Linear	Destination pitch
B4	OFFSET	Linear	Screen origin (0,0)

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C0000130	CONVSP	XY-to-linear conversion (source pitch)
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

Window Checking Window checking **cannot** be used with this instruction. The W bits are ignored.

Pixel Processing Pixel processing **cannot** be used with this instruction.

Transparency Transparency **cannot** be used with this instruction.

Plane Mask The plane mask is enabled for this instruction.

Words 1

Machine States 6,9

Status Bits
N Undefined
C Undefined
Z Undefined
V Set to 1 if the pixel is 1, set to 0 if the pixel is 0.

Examples These PIXT examples use the following implied operand setup.

Register File B:		**I/O Registers:**
DPTCH (B3)	= >800	CONVSP = >0014
OFFSET (B4)	= >00000000	

Assume that memory address @>20500 contains >CF3F before instruction execution.

```
PIXT   *A0.XY,A1
```

Before			**After**
A0	**PSIZE**	**PMASK**	**A1**
>0040 0500	>0001	>0000	>0000 0001
>0040 0500	>0001	>FFFF	>0000 0000
>0040 0280	>0002	>0000	>0000 0003
>0040 0280	>0002	>AAAA	>0000 0001
>0040 0140	>0004	>0000	>0000 000F
>0040 0140	>0004	>9999	>0000 0006
>0040 00A0	>0008	>0000	>0000 003F
>0040 00A0	>0008	>8989	>0000 0036
>0040 0050	>0010	>0000	>0000 CF3F
>0040 0050	>0010	>7310	>0000 8C2F

Note:

The XY addresses stored in register A1 in these examples translate to the linear memory address >20500. The pitch of the source was not changed for any of these examples.

PIXT Pixel Transfer - *Indirect XY to Indirect XY* PIXT

Syntax **PIXT** **<Rs>*.**XY**, **<Rd>*.**XY**

Execution (pixel)*Rs.XY → (pixel)*Rd.XY

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	Rs				R	Rd			

Operands

***Rs.XY** *Source register indirect XY format.* The source pixel is at the XY memory address contained in the specified register. The X value occupies the 16 LSBs of the register and the Y value occupies the 16 MSBs.

***Rd.XY** *Destination register indirect XY format.* The destination location is the XY address contained in the specified register. The X value occupies the 16 LSBs of the register and the Y value occupies the 16 MSBs.

Description PIXT transfers a pixel from the XY memory address contained in the source register to the XY memory address contained in the destination register. The source and destination registers must be in the same register file.

Implied Operands

B File Registers			
Register	**Name**	**Format**	**Description**
B1	SPTCH	Linear	Source pitch
B3	DPTCH	Linear	Destination pitch
B4	OFFSET	Linear	Screen origin (0,0)
B5	WSTART	XY	Window starting corner
B6	WEND	XY	Window ending corner

I/O Registers		
Address	**Name**	**Description and Elements (Bits)**
>C00000B0	CONTROL	PP– Pixel processing operations (22 options) W – Window clipping or pick operation T – Transparency operation
>C0000130	CONVSP	XY-to-linear conversion (source pitch)
>C0000140	CONVDP	XY-to-linear conversion (destination pitch)
>C0000150	PSIZE	Pixel size (1,2,4,8,16)
>C0000160	PMASK	Plane mask – pixel format

Window Checking Window clipping can be selected by setting the W bits in the CONTROL I/O register to 2 or 3. Pick can be selected by setting the W bits to 1. The WSTART and WEND registers define the window in XY-coordinate space. If window clipping or pick is not selected, then the WSTART and WEND registers are ignored. The default case at reset is no window clipping. For more information, see Section 7.10, Window Checking, on page 7-25.

Pixel Processing The PP field of the CONTROL I/O register specifies the pixel processing operation to be applied to pixels as they are transferred to the destination array. The default case at reset is the pixel processing *replace* operation. For more information, see Section 7.7, Pixel Processing, on page 7-15.

Transparency Transparency can be enabled for this instruction by setting the T bit in the CONTROL I/O register to 1. The TMS34010 checks for 0 (transparent) pixels *after* it processes the source data. At reset, the default case for transparency is *off*.

Plane Mask The plane mask is enabled for this instruction.

Words 1

Machine States

Pixel Processing Operation								Window Violation		
PSIZE	Replace	Boolean	ADD	ADDS	SUB	SUBS	MIN/MAX	W=1	W=2	W=3
1,2,4,8	7+(3),13	9+(3),15	9+(3),15	10+(3),16	10+(3),16	11+(3),17	10+(3),16	–	8,11	6,9
16	7+(1),11	9+(1),13	9+(1),13	10+(1),14	10+(1),14	11+(1),15	10+(1),14	–	8,11	6,9

Status Bits
- **N** Unaffected
- **C** Unaffected
- **Z** Unaffected
- **V** *1* if window clipping enabled and window violation occurs, *0* if no window violation occurs. Unaffected if window clipping is not enabled.

Examples These PIXT examples use the following implied operand setup.

Register File B:
SPTCH (B1) = >800
DPTCH (B3) = >800
OFFSET (B4) = >00000000
WSTART (B5) = >00300020
WEND (B6) = >00500142

I/O Registers:
CONVSP = >0014
CONVDP = >0014

```
PIXT   *A0.XY,*A1.XY
```

	Before								After	
	A0	A1	@>20500	PSIZE	PP	W	T	PMASK	@>20500	@>20510
1)	>0040 0500	>0040 0508	>000F	>0001	00000	00	0	>0000	>010F	xxxx
1)	>0040 0280	>0040 0284	>000F	>0002	00000	00	0	>0000	>030F	xxxx
1)	>0040 0140	>0040 0142	>000F	>0004	00000	00	0	>0000	>0F0F	xxxx
1)	>0040 00A0	>0040 00A1	>00EF	>0008	00000	00	0	>0000	>EFEF	xxxx
1)	>0040 0050	>0040 0051	>CDEF	>0010	00000	00	0	>0000	>CDEF	>CDEF
2)	>0040 0140	>0040 0142	>0306	>0004	01010	00	0	>0000	>0506	xxxx
3)	>0040 0140	>0040 0142	>0106	>0004	00001	00	0	>0000	>0006	xxxx
4)	>0040 0140	>0040 0142	>0106	>0004	10001	00	1	>0000	>0106	xxxx
5)	>0040 0140	>0040 0142	>0006	>0004	00000	00	0	>AAAA	>0406	xxxx
6)	>0040 0140	>0040 0142	>0006	>0004	00000	11	0	>0000	>0606	xxxx
7)	>0040 0140	>0040 0143	>0006	>0004	00000	11	0	>0000	>0006	xxxx
8)	>0040 0140	>0040 0143	>0006	>0004	00000	10	0	>0000	>0006	xxxx

XY Address in A0 = Linear Address >20500

Notes:

1) S replaces D
2) (S XOR D) replaces D
3) (S AND D) = 0, transparency is off, D is replaced
4) (S + D) = 0, transparency is on, D not replaced
5) S replaces unmasked bits of D
6) Window Option = 3, D inside window, S replaces D
7) Window Option = 3, D outside window, D not replaced, V bit set in status register
8) Window Option = 2, D outside window, D not replaced, WV interrupt generated, V bit set in status register

Syntax POPST

Execution *SP+ → ST

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0

Description POPST pops the status register from the stack and increments the SP by 32 after the status register is removed from the stack.

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

N	C	Z	V	Res	PBX	Res	IE	Reserved	FE1	FS1	FE0	FS0

Status Register

Words 1

Machine States 8,11 (SP aligned)
10,13 (SP nonaligned)

Status Bits

- **N** Set from bit 31 of stack status.
- **C** Set from bit 30 of stack status.
- **Z** Set from bit 29 of stack status.
- **V** Set from bit 28 of stack status.
- **IE** Set from bit 21 of stack status.

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>0FF0 0000	>0010
>0FF0 0010	>C000

Code	Before	After	
	SP	**ST**	**SP**
POPST	>0FF0 0000	>C000 0010	>0FF0 0020

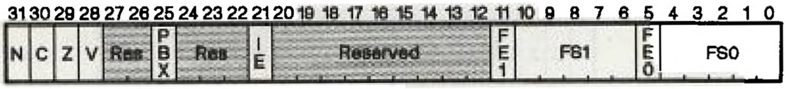
Syntax **PUSHST**

Execution ST → -*SP

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0

Description PUSTST pushes the status register onto the stack and then decrements the SP by 32.



Status Register

Words 1

Machine States
2+(3),8 (SP aligned)
2+(8),13 (SP nonaligned)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Example

Code	Before		After
	SP	ST	SP
`PUSHST`	>0FF0 0020	>C000 0010	>0FF0 0000

Memory will contain the following values after instruction execution:

Address	Data
>0FF0 0010	>0010
>0FF0 0020	>C000

PUTST Copy Register into Status

Syntax **PUTST** *<Rs>*

Execution (Rs) → ST

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	0	1	R	Rs			

Description PUTST copies the contents of the specified register into the status register.

31	30	29	28	27–26	25	24–22	21	20–12	11	10–6	5	4–0
N	C	Z	V	Res	PBX	Res	IE	Reserved	FE1	FS1	FE0	FS0

Status Register

Words 1

Machine States 3,6

Status Bits

- **N** Set to value of bit 31 in source register
- **C** Set to value of bit 30 in source register
- **Z** Set to value of bit 29 in source register
- **V** Set to value of bit 28 in source register
- **IE** Set to value of bit 21 in source register

Example

Code	Before		After
	A0	ST	ST
`PUTST A0`	>C000 0010	>xxxx xxxx	>C000 0010

Syntax RETI

Execution
*SP+ → ST
*SP+ → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0

Description RETI returns from an interrupt routine. It pops the status register and then the program counter from the stack. Execution then continues according to the values loaded.

The stack is located in external memory and the top is indicated by the stack pointer (SP). The stack grows in the direction of decreasing linear address. The ST and PC are popped from the stack and the SP is incremented by 32 after each register is removed from the stack.

> **Note:**
>
> If the PBX status bit is set in the restored ST value, then the bit is cleared and a PIXBLT or FILL will be resumed, depending on the values stored in the B-file registers.
>
> The CONTROL register and any B-file registers modified by an interrupt routine should be restored before RETI is executed. Otherwise, interrupted PIXBLT and FILL instructions may not resume execution correctly.

Words 1

Machine States
11,14 (aligned stack)
15,18 (nonaligned stack)

Status Bits
N Copy of corresponding bit in stack location
C Copy of corresponding bit in stack location
Z Copy of corresponding bit in stack location
V Copy of corresponding bit in stack location
IE Copy of corresponding bit in stack location

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>0CCC 0000	>0010
>0CCC 0010	>C000
>0CCC 0020	>FFF0
>0CCC 0030	>0044

Code	Before		After	
	SP	ST	PC	SP
RETI	>0CCC 0000	>C000 0010	>0044 FFF0	>0CCC 0040

Syntax **RETS** [<*N*>]

Execution *SP → PC (N defaults to 0)
(SP) + 32+ (16N) → SP

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	0	1	1	N				

Fields **N** Optional stack pointer adjustment (0 to 31 words)

Description RETS returns from a subroutine by popping the program counter from the stack and incrementing the stack pointer by *N* +2 words. If *N* is specified, the stack pointer is incremented by 32 + 16*N*. If *N* is not specified, the stack is incremented by 32. Execution then continues according to the PC value loaded.

Words 1

Machine States 7,10 (Aligned stack)
9,12 (Unaligned stack)

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples Assume that memory contains the following values before instruction execution:

Address	Data
>0FF0 0000	>FFF0
>0FF0 0010	>0001

Code	Before SP	After PC	After SP
RETS	>0FF0 0000	>0001 FFF0	>0FF0 0020
RETS 1	>0FF0 0000	>0001 FFF0	>0FF0 0030
RETS 2	>0FF0 0000	>0001 FFF0	>0FF0 0040
RETS 16	>0FF0 0000	>0001 FFF0	>0FF0 0120
RETS 31	>0FF0 0000	>0001 FFF0	>0FF0 0210

Syntax **REV** *<Rd>*

Execution Revision numer → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	R	Rd			

Description REV stores the revision number of the TMS340 family device in the destination register. The revision number information is stored in the following format:

31	30	29	...	4	3	2	1	0
0	0	0	...	0	1	Reserved		

Words 1

Machine States 1,4

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before	After
	A1	A1
REV A1	>FFFF FFFF	>0000 0008

RL Rotate Left - *Constant* RL

Syntax RL *<K>,<Rd>*

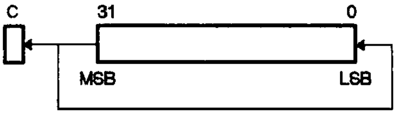
Execution (Rd) rotated left by K → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	K					R	Rd			

Operands K is a rotate count from 0 to 31.

Description RL rotates the destination register contents by left the number of bits specified by K. This is a circular rotate so that bits shifted out the MSB are shifted into the LSB.



The left rotate count is contained in the 5-bit K field of the instruction word. The assembler will only accept absolute expressions as valid K operand values. If the value specified is greater than 31, the assembler will issue a warning and set the value of the K field equal to the five LSBs of the K operand value specified.

The rotate count of 0 can be used to clear the carry and test a register for 0 simultaneously.

Words 1

Machine States 1,4

Status Bits
- **N** Unaffected
- **C** Set to value of last bit rotated out, *0* for rotate count of 0.
- **Z** *1* if result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before	After	
	A1	NCZV	A1
`RL 0,A1`	>0000 000F	x00x	>0000 000F
`RL 1,A1`	>F000 0000	x10x	>E000 0001
`RL 4,A1`	>F000 0000	x10x	>0000 000F
`RL 5,A1`	>F000 0000	x00x	>0000 001E
`RL 30,A1`	>F000 0000	x10x	>3C00 0000
`RL 5,A1`	>0000 0000	x01x	>0000 0000

Syntax RL *<Rs>,<Rd>*

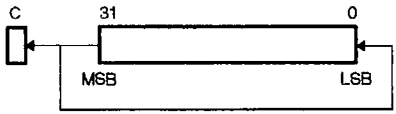
Execution (Rd) rotated left by Rs → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	Rs				R	Rd			

Operands **Rs** The five LSBs of the source register specify the left rotate count (a value from 0 to 31). The 27 MSBs are ignored.

Description RL rotates the destination register contents left by the number of bits specified. This is a circular rotate, so that bits shifted out of the MSB are shifted into the LSB.



Note that the you must designate Rs with a keyword or symbol which has been defined to be a register, for instance A9. Otherwise, the assembler will use the `RL K,Rd` instruction.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
N Unaffected
C Set to value of last bit rotated out, *0* for rotate count of 0.
Z *1* if result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before 5 LSBs A0	Before A1	After NCZV	After A1
`RL A0,A1`	00000	>0000 000F	x00x	>0000 000F
`RL A0,A1`	00100	>F000 0000	x10x	>0000 000F
`RL A0,A1`	00101	>F000 0000	x00x	>0000 001E
`RL A0,A1`	11111	>F000 0000	x00x	>7800 0000
`RL A0,A1`	xxxxx	>0000 0000	x01x	>0000 0000

Syntax **SETC**

Execution 1 → C

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0

Description SETC sets the carry bit (C) in the status register to 1. The rest of the status register is unaffected.

This instruction is useful for returning a true/false value (in the carry bit) from a subroutine without using a general-purpose register.

Words 1

Machine States 1,4

Status Bits
N Unaffected
C *1*
Z Unaffected
V Unaffected

Examples

Code	Before ST	Before NCZV	After ST	After NCZV
SETC	>0000 0000	0000	>4000 0000	0100
SETC	>B000 0010	1011	>F000 0010	1111
SETC	>4000 001F	0100	>4000 001F	0100

Syntax **SETF** *<FS>,<FE>*[,*<F>*]

Execution (FS, FE) → ST

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	F	1	0	1	FE	FS				

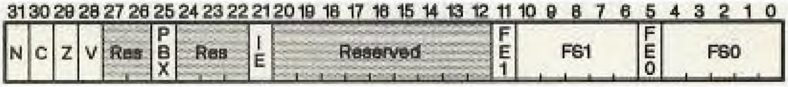
Operands

FS is the field size to be stored in status register (1–32).

FE is the field extend to be stored in status register – 0 for zero extend, 1 for sign extend.

F is an optional operand; it defaults to 0.
F=0 selects FS0, FE0 to be altered.
F=1 selects FS1, FE1 to be altered.

Description SETF loads the values specified for the field size (FS) and the field extension (FE) into the status register. The rest of ST is unchanged. The F bit specifies whether the Field 0 or Field 1 parameters are to be set. FS can be in the range 1–32, FE is either 0 or 1, and F is optional. If F is not specified, it defaults to 0. An FS of 0 in the opcode corresponds to a field size of 32. This instruction is used to set either of the two sets of field move parameters in the status register. These determine the field size for MOVE field instructions and the field-extension rule for MOVE into a register. Either set of parameters can be chosen by an individual MOVE instruction, by specifying the F parameter.



Status Register

Words 1

Machine States 1,4 for F=0
2,5 for F=1

Status Bits
N Unaffected
C Unaffected
Z Unaffected
V Unaffected

Examples

Code	Before	After
	ST	ST
`SETF 32,0,0`	>xxxx x000	>xxxx x000
`SETF 32,1,0`	>xxxx x000	>xxxx x020
`SETF 31,1,0`	>xxxx x000	>xxxx x03F
`SETF 16,0,0`	>xxxx x000	>xxxx x010
`SETF 32,0,1`	>xxxx x000	>xxxx x000
`SETF 32,1,1`	>xxxx x000	>xxxx x800
`SETF 31,1,1`	>xxxx x000	>xxxx xFC0
`SETF 16,0,1`	>xxxx x000	>xxxx x400

SEXT — Sign Extend to Long

Syntax **SEXT** *<Rd>*[,*<F>*]

Execution (field)Rd → (sign-extended field) Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	F	1	0	0	0	R	Rd			

Operands

F Is an optional operand; it defaults to 0
0 selects FS0 for the field size
1 selects FS1 for the field size

Description SEXT sign extends the right-justified field contained in the destination register by copying the MSB of the field data into all the nonfield bits of the destination register. The field size for the sign extension is specified by the FS0 or FS1 bits in the status register, depending on the F bit specified.

Words 1

Machine States 3,6

Status Bits

N *1* if the result is negative, *0* otherwise.
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before FS0/1	Before A0	After NCZV	After A0
`SEXT A0,0`	17/x	>0000 8000	0x0x	>0000 8000
`SEXT A0,0`	16/x	>0000 8000	1x0x	>FFFF 8000
`SEXT A0,0`	15/x	>0000 8000	0x1x	>0000 0000
`SEXT A0,1`	x/17	>0000 8000	0x0x	>0000 8000
`SEXT A0,1`	x/16	>0000 8000	1x0x	>FFFF 8000
`SEXT A0,1`	x/15	>0000 8000	0x1x	>0000 0000

Syntax SLA *<K>,<Rd>*

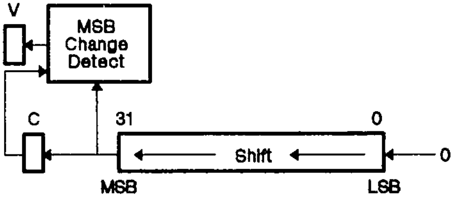
Execution (Rd) shifted left by K → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	K					R	Rd			

Operands K is a shift value from 0 to 31.

Description SLA shifts the destination register contents left by the number of bits specified. As shown in the diagram, zeros are shifted into the least significant bits. The last bit shifted out of the destination register is shifted into the carry bit. If either the sign bit (N) or any of the bits shifted out of the register differ from the original sign bit, the overflow bit (V) is set.



The left shift count is contained in the 5-bit K field of the instruction word. The assembler accepts only absolute expressions as valid K operand values. SLA executes slower than SLL because overflow detection. If the value specified is greater than 31, the assembler issues a warning and sets the value of the K field equal to the five LSBs of the K operand value specified.

Words 1

Machine States 3,6

Status Bits
- **N** *1* if the result is negative, *0* otherwise.
- **C** Set to the value of last bit shifted out, *0* for shift count of 0.
- **Z** *1* if a 0 result generated, *0* otherwise.
- **V** *1* if the MSB changes during shift operation, *0* otherwise.

Examples

Code	Before A1	After A1	NCZV
`SLA 0,A1`	>3333 3333	>3333 3333	0000
`SLA 0,A1`	>CCCC CCCC	>CCCC CCCC	1000
`SLA 1,A1`	>CCCC CCCC	>9999 9998	1100
`SLA 2,A1`	>3333 3333	>CCCC CCCC	1001
`SLA 2,A1`	>CCCC CCCC	>3333 3330	0101
`SLA 3,A1`	>CCCC CCCC	>6666 6660	0001
`SLA 5,A1`	>CCCC CCCC	>9999 9980	1101
`SLA 30,A1`	>CCCC CCCC	>0000 0000	0111
`SLA 31,A1`	>CCCC CCCC	>0000 0000	0011
`SLA 31,A1`	>0000 0000	>0000 0000	0010

Syntax **SLA** *<Rs>,<Rd>*

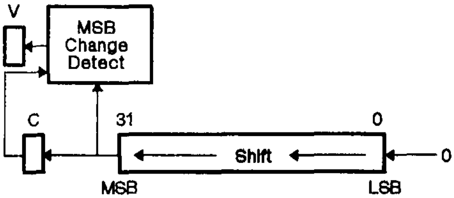
Execution (Rd) shifted left by (Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	Rs				R	Rd			

Operands **Rs** The five LSBs of the source register specify the left-shift count (a value from 0 to 31). The 27 MSBs are ignored.

Description SLA shifts the destination register contents left by the number of bits specified the source register. The last bit shifted out of the destination register is shifted into the carry bit. If either the sign bit (N) or any of the bits shifted out of the register differ from the original sign bit, the overflow bit (V) is set.



The left shift count is specified by the five LSBs of the source register.

Note that you must designate Rs with a keyword or symbol which has been defined to be a register, for instance A9. Otherwise, the assembler will use the `SLA K,Rd` instruction. SLA executes slower than SLL because the overflow detection. The source and destination registers must be in the same register file.

Words 1

Machine States 3,6

Status Bits
N *1* if the result is negative, *0* otherwise.
C Set to value of last bit shifted out, *0* for shift count of 0.
Z *1* if the result is 0, *0* otherwise.
V *1* if the MSB changes during shift operation, *0* otherwise.

Examples

Code	Before 5 LSBs A0	Before A1	After A1	NCZV
`SLA A0,A1`	00000	>3333 3333	>3333 3333	0000
`SLA A0,A1`	00000	>CCCC CCCC	>CCCC CCCC	1000
`SLA A0,A1`	00001	>CCCC CCCC	>9999 9998	1100
`SLA A0,A1`	00010	>3333 3333	>CCCC CCCC	1001
`SLA A0,A1`	00010	>CCCC CCCC	>3333 3330	0101
`SLA A0,A1`	00011	>CCCC CCCC	>6666 6660	0001
`SLA A0,A1`	00101	>CCCC CCCC	>9999 9980	1101
`SLA A0,A1`	11110	>CCCC CCCC	>0000 0000	0111
`SLA A0,A1`	11111	>CCCC CCCC	>0000 0000	0011
`SLA A0,A1`	11111	>0000 0000	>0000 0000	0010

Syntax **SLL** *<K>,<Rd>*

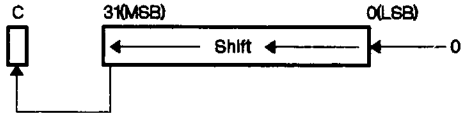
Execution (Rd) shifted left by K → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	K					R	Rd			

Operands **K** is a shift value from 0 to 31.

Description SLL shifts the destination register contents left by the number of bits specified. The last bit shifted out of the destination register is shifted into the carry bit. Zeros are shifted into the least significant bits. This instruction differs from the SLA instruction only in its effect on the overflow (V) bit.



The left shift count is contained in the 5-bit K field of the instruction word. The assembler will only accept absolute expressions as valid K operand values. If the value specified is greater than 31, the assembler will issue a warning and set the value of the K field equal to the five LSBs of the K operand value specified.

Words 1

Machine States 1,4

Status Bits
N Unaffected
C *1* to the value of last bit shifted out, *0* for shift count of 0.
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before A1	After A1	NCZV
SLL 0,A1	>0000 0000	>0000 0000	x01x
SLL 0,A1	>8888 8888	>8888 8888	x00x
SLL 1,A1	>8888 8888	>1111 1110	x10x
SLL 4,A1	>8888 8888	>8888 8880	x00x
SLL 30,A1	>FFFF FFFC	>0000 0000	x11x
SLL 31,A1	>FFFF FFFC	>0000 0000	x01x

Shift Left Logical - *Register*

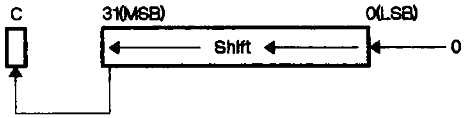
Syntax **SLL** *<Rs>,<Rd>*

Execution (Rd) shifted left by (Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	Rs				R	Rd			

Description SLL shifts the destination register contents left by the number of bits specified in the source register. The last bit shifted out of the destination register is shifted into the carry bit. Zeros are shifted into the least significant bits. The left shift count is specified by the five LSBs of the source register. This instruction differs from the SLA instruction only in its effect on the overflow (V) bit.



Note that you must designate Rs with a keyword or symbol which has been defined to be a register, for instance A9. Otherwise, the assembler will use the `SLA K,Rd` instruction.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

N Unaffected
C Set to the value of last bit shifted out, *0* for shift value of 0.
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before 5 LSBs A0	Before A1	After A1	NCZV
`SLL A0,A1`	00000	>0000 0000	>0000 0000	x01x
`SLL A0,A1`	00000	>8888 8888	>8888 8888	x00x
`SLL A0,A1`	00001	>8888 8888	>1111 1110	x10x
`SLL A0,A1`	00100	>8888 8888	>8888 8880	x00x
`SLL A0,A1`	11110	>FFFF FFFC	>0000 0000	x11x
`SLL A0,A1`	11111	>FFFF FFFC	>0000 0000	x01x

Syntax SRA *<K>,<Rd>*

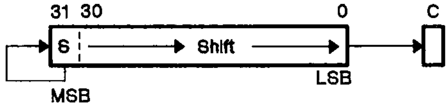
Execution (Rd) shifted right by K → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	2s Complement of K					R	Rd			

Operands K is a shift count from 0 to 31.

Description SRA shifts the destination register contents right by the number of bits specified. The last bit shifted out of the destination register is shifted into the carry bit. The sign bit (MSB) is extended into the most significant bits.



The 5-bit K field of the instruction opcode contains the 2's complement of the right shift count specified by the K operand. The assembler will only accept absolute expressions for the shift operand value. If the value specified is greater than 31, the assembler will issue a warning and set the value of the K field of the instruction opcode equal to the 2's complement of the five LSBs of the specified operand value.

Words 1

Machine States 1,4

Status Bits
- **N** *1* if the result is negative, *0* otherwise.
- **C** Set to the value of last bit shifted out, *0* for shift count of 0.
- **Z** *1* if the result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before A1	After A1	NCZV
`SRA 0,A1`	>0000 0000	>0000 0000	001x
`SRA 0,A1`	>FFFF 0000	>FFFF 0000	100x
`SRA 8,A1`	>7FFF 0000	>007F FF00	000x
`SRA 8,A1`	>FFFF 0000	>FFFF FF00	100x
`SRA 30,A1`	>7FFF 0000	>0000 0001	010x
`SRA 31,A1`	>7FFF 0000	>0000 0000	011x
`SRA 31,A1`	>FFFF 0000	>FFFF FFFF	110x

Syntax **SRA** *<Rs>,<Rd>*

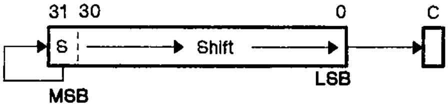
Execution (Rd) shifted right by -(Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	Rs				R	Rd			

Operands **Rs** The 2's complement of the source register's five LSBs specify a shift count from 0–31 bits. The 27 MSBs are ignored.

Description SRA shifts the destination register contents right by the number of bits specified in the source register. The last bit shifted out of the destination register is shifted into the carry bit. The sign bit (MSB) is extended into the most significant bits.



> **Note:**
>
> The five LSBs of the source register contain the 2's complement of the right shift count.

You must specify Rs with a keyword or a symbol which has been defined to be a register, for instance A9. Otherwise, the assembler will use the `SRA K,Rd` instruction. The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
- **N** *1* if the result is negative, *0* otherwise.
- **C** Set to the value of last bit shifted out, *0* for shift count of 0.
- **Z** *1* if the result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before 5 LSBs A0	Before A1	After A1	After NCZV
`SRA A0,A1`	00000	>0000 0000	>0000 0000	001x
`SRA A0,A1`	00000	>FFFF 0000	>FFFF 0000	100x
`SRA A0,A1`	11111	>7FFF 0000	>3FFF 8000	000x
`SRA A0,A1`	11111	>FFFF 0000	>FFFF 8000	100x
`SRA A0,A1`	11000	>7FFF 0000	>007F FF00	000x
`SRA A0,A1`	11000	>FFFF 0000	>FFFF FF00	100x
`SRA A0,A1`	00010	>7FFF 0000	>0000 0001	010x
`SRA A0,A1`	00001	>7FFF 0000	>0000 0000	011x
`SRA A0,A1`	00001	>FFFF 0000	>FFFF FFFF	110x

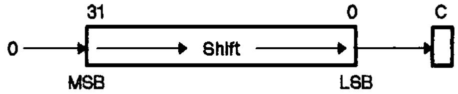
Syntax	**SRL** *<K>,<Rd>*
Execution	(Rd) shifted right by K → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	2s Complement of K					R	Rd			

Operands **K** is a shift value from 0 to 31.

Description SRL shifts the destination register contents right by the number of bits specified. The last bit shifted out of the destination register is shifted into the carry bit. Zeros are shifted into the most significant bits.



The 5-bit K field of the instruction opcode contains the 2's complement of the right shift count specified by the K operand. The assembler accepts only absolute expressions for the shift operand value. If the specified value is greater than 31, the assembler issues a warning and set the value of the K field of the instruction opcode equal to the 2's complement of the five LSBs of the specified operand value.

Words 1

Machine States 1,4

Status Bits

- **N** Unaffected
- **C** Set to the value of last bit shifted out, *0* for shift count of 0.
- **Z** *1* if the result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before A1	After A1	NCZV
`SRL 0,A1`	>0000 0000	>0000 0000	x01x
`SRL 0,A1`	>7FFF FFFF	>7FFF FFFF	x00x
`SRL 1,A1`	>7FFF FFFF	>3FFF FFFF	x10x
`SRL 8,A1`	>7FFF 0000	>007F FF00	x00x
`SRL 30,A1`	>7FFF 0000	>0000 0001	x10x
`SRL 31,A1`	>7FFF 0000	>0000 0000	x11x
`SRL 31,A1`	>3FFF 0000	>0000 0000	x01x

SRL Shift Right Logical - *Register* SRL

Syntax **SRL** *<Rs>,<Rd>*

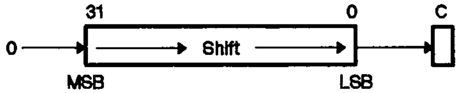
Execution (Rd) shifted right by -(Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	Rs				R	Rd			

Operands **Rs** The 2's complement of the source register's five LSBs specify a shift count from 0–31 bits. The 27 MSBs are ignored.

Description SRL shifts the destination register contents right by the number of bits specified. The last bit shifted out of the destination register is shifted into the carry bit. Zeros are shifted into the most significant bits.



> **Note:** The five LSBs of the source register contain the 2's complement of the right shift count.

You must specify Rs with a keyword or symbol which has been defined to be a register, for instance A9. Otherwise, the assembler will use the SRL K,Rd instruction. The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
- **N** Unaffected
- **C** Set to the value of last bit shifted out, *0* for shift count of 0.
- **Z** *1* if the result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before 5 LSBs A0	Before A1	After A1	NCZV
SRL A0,A1	00000	>0000 0000	>0000 0000	x01x
SRL A0,A1	00000	>7FFF FFFF	>7FFF FFFF	x00x
SRL A0,A1	11111	>7FFF FFFF	>3FFF FFFF	x10x
SRL A0,A1	11000	>7FFF 0000	>007F FF00	x00x
SRL A0,A1	00010	>7FFF 0000	>0000 0001	x10x
SRL A0,A1	00001	>7FFF 0000	>0000 0000	x11x
SRL A0,A1	00001	>3FFF 0000	>0000 0000	x01x

Syntax SUB *<Rs>,<Rd>*

Execution (Rd) - (Rs) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	Rs				R	Rd			

Operands

Rs contains the 32-bit subtrahend.

Rd contains the 32-bit minuend.

Description SUB subtracts the contents of the source register from the contents of the destination register; the result is stored in the destination register. Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the SUBB instruction.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A0	Before A1	After NCZV	After A0
`SUB A1,A0`	>7FFF FFF2	>7FFF FFF1	0000	>0000 0001
`SUB A1,A0`	>7FFF FFF2	>7FFF FFF2	0010	>0000 0000
`SUB A1,A0`	>7FFF FFF1	>7FFF FFF2	1100	>FFFF FFFF
`SUB A1,A0`	>7FFF FFF1	>FFFF FFFF	0100	>7FFF FFF2
`SUB A1,A0`	>7FFF FFFF	>FFFF FFFF	1101	>8000 0000
`SUB A1,A0`	>FFFF FFFD	>FFFF FFFF	1100	>FFFF FFFE
`SUB A1,A0`	>FFFF FFFD	>FFFF FFFD	0010	>0000 0000
`SUB A1,A0`	>FFFF FFFE	>FFFF FFFD	0000	>0000 0001
`SUB A1,A0`	>FFFF FFFF	>0000 0001	1000	>FFFF FFFE
`SUB A1,A0`	>8000 0000	>0000 0001	0001	>7FFF FFFF

Syntax **SUBB** *<Rs>*,*<Rd>*

Execution (Rd) - (Rs) - (C) → Rd (C acts as borrow)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	Rs				R	Rd			

Operands **Rs** contains the 32-bit subtrahend.

Rd contains the 32-bit minuend.

Description SUBB subtracts both the contents of the source register and the carry bit from the contents of the destination register; the result is stored in the destination register. This instruction can be used with the SUB, SUBK, and SUBI instructions for extended-precision arithmetic.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before C	Before A0	Before A1	After NCZV	After A0
`SUBB A1,A0`	0	>0000 0002	>0000 0001	0000	>0000 0001
`SUBB A1,A0`	1	>0000 0002	>0000 0001	0010	>0000 0000
`SUBB A1,A0`	0	>0000 0002	>0000 0002	0010	>0000 0000
`SUBB A1,A0`	1	>0000 0002	>0000 0002	1100	>FFFF FFFF
`SUBB A1,A0`	0	>0000 0002	>0000 0003	1100	>FFFF FFFF
`SUBB A1,A0`	0	>7FFF FFFE	>FFFF FFFF	0100	>7FFF FFFF
`SUBB A1,A0`	0	>7FFF FFFE	>FFFF FFFE	1101	>8000 0000
`SUBB A1,A0`	1	>7FFF FFFE	>FFFF FFFE	0100	>7FFF FFFF
`SUBB A1,A0`	0	>FFFF FFFE	>FFFF FFFF	1100	>FFFF FFFF
`SUBB A1,A0`	0	>FFFF FFFE	>FFFF FFFE	0010	>0000 0000
`SUBB A1,A0`	1	>FFFF FFFE	>FFFF FFFE	1100	>FFFF FFFF
`SUBB A1,A0`	0	>FFFF FFFE	>FFFF FFFD	0000	>0000 0001
`SUBB A1,A0`	1	>FFFF FFFE	>FFFF FFFD	0010	>0000 0000
`SUBB A1,A0`	0	>8000 0001	>0000 0001	1000	>8000 0000
`SUBB A1,A0`	1	>8000 0001	>0000 0001	0001	>7FFF FFFF
`SUBB A1,A0`	0	>8000 0001	>0000 0002	0001	>7FFF FFFF

Syntax **SUBI** *<IW>*,*<Rd>* [,**W**]

Execution (Rd) - IW → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	1	1	R	Rd			
~IW															

Operands **IW** is a signed 16-bit immediate value.

Description SUBI subtracts the sign-extended, 16-bit immediate value from the contents of the destination register; the result is stored in the destination register.

The assembler will use the short form if the immediate value has been previously defined and is in the range -32,768 ≤ IW ≤ 32,767. You can force the assembler to use the short form by by following the register specification with **,W**:

```
SUBI  IW,Rd,W
```

The assembler will truncate any upper bits and issue an appropriate warning message. Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the SUBB instruction.

Words 2

Machine States 2,8

Status Bits
N *1* if the result is negative, *0* otherwise.
C *1* if a borrow is generated, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before A0	After A0	NCZV
SUBI 32765,A0	>0000 7FFE	>0000 0001	0000
SUBI 32766,A0	>0000 7FFE	>0000 0000	0010
SUBI 32767,A0	>0000 7FFE	>FFFF FFFF	1100
SUBI 32766,A0	>8000 7FFE	>8000 0000	1000
SUBI 32767,A0	>8000 7FFE	>7FFF FFFF	0001
SUBI -32766,A0	>FFFF 8001	>FFFF FFFF	1100
SUBI -32767,A0	>FFFF 8001	>0000 0000	0010
SUBI -32768,A0	>FFFF 8001	>0000 0001	0000
SUBI -32767,A0	>7FFF 8000	>7FFF FFFF	0100
SUBI -32768,A0	>7FFF 8000	>8000 0000	1101

Syntax **SUBI** *<IL>*,*<Rd>*[,**L**]

Execution (Rd) - IL → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	0	0	R	Rd			
~IL (LSW)															
~IL (MSW)															

Operands **IL** is a signed 32-bit immediate value.

Description SUBI subtracts the signed 32-bit immediate value from the contents of the destination register; the result is stored in the destination register. The assembler will use this opcode if it cannot use the `SUBI IW,Rd` opcode, or if the long opcode is forced by following the register specification with **,L**:

```
SUBI  IL,Rd,L
```

Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the SUBB instruction.

Words 3

Machine States 3,12

Status Bits
N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code		Before A0	After A0	NCZV
SUBI	2147483647,A0	>7FFF FFFF	>0000 0000	0010
SUBI	32768,A0	>0000 8001	>0000 0001	0000
SUBI	32769,A0	>0000 8001	>0000 0000	0010
SUBI	32770,A0	>0000 8001	>FFFF FFFF	1100
SUBI	32768,A0	>8000 8000	>8000 0000	1000
SUBI	32769,A0	>8000 8000	>7FFF FFFF	0001
SUBI	-2147483648,A0	>8000 0000	>0000 0000	0010
SUBI	-32769,A0	>FFFF 7FFE	>FFFF FFFF	1100
SUBI	-32770,A0	>FFFF 7FFE	>0000 0000	0010
SUBI	-32771,A0	>FFFF 7FFE	>0000 0001	0000
SUBI	-32770,A0	>7FFF 7FFD	>7FFF FFFF	0100
SUBI	-32771,A0	>7FFF 7FFD	>8000 0000	1101

Syntax SUBK *<K>*,*<Rd>*

Execution (Rd) - K → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	K					R	Rd			

Operands K is a constant from 1 to 32.

Description SUBK subtracts the 5-bit constant from the contents of the destination register; the result is stored in the destination register. The constant is treated as an unsigned number in the range 1–32, where K = 0 in the opcode corresponds to the value 32. The assembler converts the value 32 to 0. The assembler issues an error if you try to subtract 0 from a register. Multiple-precision arithmetic can be accomplished by using this instruction in conjunction with the SUBB instruction.

Words 1

Machine States 1,4

Status Bits

N *1* if the result is negative, *0* otherwise.
C *1* if there is a borrow, *0* otherwise.
Z *1* if the result is 0, *0* otherwise.
V *1* if there is an overflow, *0* otherwise.

Examples

Code	Before	After	
	A0	A0	NCVZ
`SUBK 5,A0`	>0000 0009	>0000 0004	0000
`SUBK 9,A0`	>0000 0009	>0000 0000	0010
`SUBK 32,A0`	>0000 0009	>FFFF FFE9	1100
`SUBK 1,A0`	>8000 0000	>7FFF FFFF	0001

Syntax **SUBXY** *<Rs>,<Rd>*

Execution
(Rd**X**) - (Rs**X**) → Rd**X**
(Rd**Y**) - (Rs**Y**) → Rd**Y**

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	Rs				R	Rd			

Description SUBXY subtracts the source X and Y values individually from the destination X and Y values; the result is stored in the destination register.

This instruction can be used for manipulating XY addresses and is particularly useful for incremental figure drawing. These addresses are stored as XY pairs in the register file.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits
N *1* if source X field = destination X field, *0* otherwise.
C *1* if source Y field > destination Y field, *0* otherwise.
Z *1* if source Y field = destination Y field, *0* otherwise.
V *1* if source X field > destination X field, *0* otherwise.

Examples

Code	Before A0	Before A1	After A0	NCZV
SUBXY A1,A0	>0009 0009	>0001 0001	>0008 0008	0000
SUBXY A1,A0	>0009 0009	>0009 0001	>0000 0008	0010
SUBXY A1,A0	>0009 0009	>0001 0009	>0008 0000	1000
SUBXY A1,A0	>0009 0009	>0009 0009	>0000 0000	1010
SUBXY A1,A0	>0009 0009	>0000 0010	>0009 FFF9	0001
SUBXY A1,A0	>0009 0009	>0009 0010	>0000 FFF9	0011
SUBXY A1,A0	>0009 0009	>0010 0000	>FFF9 0009	0100
SUBXY A1,A0	>0009 0009	>0010 0009	>FFF9 0000	1100
SUBXY A1,A0	>0009 0009	>0010 0010	>FFF9 FFF9	0101

Syntax **TRAP** <*N*>

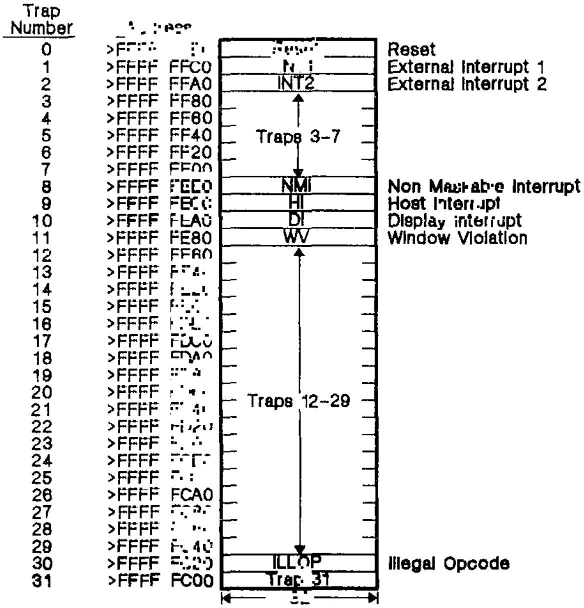
Execution
(PC) → -*SP
(ST) → -*SP
Trap Vector(N) → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	0	0	0	N				

Operands **N** is a trap number from 0 to 31.

Description TRAP executes a software interrupt. The return address (the address of next instruction) and then the status register are pushed onto the stack. The IE (interrupt enable) bit in ST is set to 0, disabling maskable interrupts, and ST is set to >0000 0010. Finally, the trap vector is loaded into the PC. The TMS34010 generates the trap vector addresses as shown below:



The stack is located in external memory and the top is indicated by the stack pointer (SP). The stack grows in the direction of decreasing linear address. The PC and ST are pushed on the stack MSW first, and the SP is predecremented before each word is loaded onto the stack.

Notes:

1. The level 0 trap differs from all other traps; it does not save the old status register or program counter. This may be useful in cases where the stack pointer is corrupted or uninitialized; such a situation could cause an erroneous write.
2. The NMI bit does not affect the operation of TRAP 8.

Words 1

Machine States 16,19 (SP aligned)
30,33 (SP nonaligned)

Status Bits
N *0*
C *0*
Z *0*
V *0*

Examples

Code	Before PC	Before SP	After PC	After SP	After ST
TRAP 0	>xxxx xxxx	>8000 0000	@FFFF FFE0	>8000 0000	>0000 0010
TRAP 1	>xxxx xxxx	>8000 0000	@FFFF FFC0	>7FFF FFC0	>0000 0010
.	.	.	.	.	.
.	.	.	.	.	.
TRAP 30	>xxxx xxxx	>8000 0000	@FFFF FC20	>7FFF FFC0	>0000 0010
TRAP 31	>xxxx xxxx	>8000 0000	@FFFF FC00	>7FFF FFC0	>0000 0010

Syntax **XOR** *<Rs>,<Rd>*

Execution (Rs) XOR (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	Rs				R	Rd			

Description XOR bitwise-exclusive-ORs the contents of the source register with the contents of the destination register; the result is stored in the destination register.

You can use this instruction to clear registers (for example, `XOR B0,B0`); the CLR instruction also supports this function.

The source and destination registers must be in the same register file.

Words 1

Machine States 1,4

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** *1* if the result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before A0	Before A1	After NCZV	After A1
`XOR A0,A1`	>FFFF FFFF	>0000 0000	xx0x	>FFFF FFFF
`XOR A0,A1`	>FFFF FFFF	>AAAA AAAA	xx0x	>5555 5555
`XOR A0,A1`	>FFFF FFFF	>FFFF FFFF	xx1x	>0000 0000

XORI Exclusive OR Immediate Value XORI

Syntax **XORI** *<IL>,<Rd>*

Execution IL XOR (Rd) → Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	1	0	R	Rd			
IL (LSW)															
IL (MSW)															

Operands **IL** is a 32-bit immediate value.

Description XORI bitwise exclusive ORs the 32-bit immediate data with the contents of the destination register; the result is stored in the destination register.

Words 3

Machine States 3,12

Status Bits

- **N** Unaffected
- **C** Unaffected
- **Z** *1* if the result is 0, *0* otherwise.
- **V** Unaffected

Examples

Code	Before	After	
	A0	**NCZV**	**A0**
XORI >FFFFFFFF,A0	>00000000	xx0x	>FFFF FFFF
XORI >FFFFFFFF,A0	>AAAAAAAA	xx0x	>5555 5555
XORI >FFFFFFFF,A0	>FFFFFFFF	xx1x	>0000 0000
XORI >00000000,A0	>00000000	xx1x	>0000 0000
XORI >00000000,A0	>FFFFFFFF	xx0x	>FFFF FFFF

Syntax **ZEXT** *<Rd>*[,*<F>*]

Execution (field) Rd → (zero-extended field) Rd

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	F	1	0	0	1	R	Rd			

Operands F is an optional parameter, it defaults to 0.
F=0 selects FS0 for the field size.
F=1 selects FS1 for the field size.

Description ZEXT zero extends the right-justified field contained in the destination register by zeroing all the nonfield bits of the destination register. The field size for the zero extension is specified by the FS0 or FS1 bits in the status register, depending on the value of F.

Words 1

Machine States 1,4

Status Bits
N Unaffected
C Unaffected
Z *1* if the result is 0, *0* otherwise.
V Unaffected

Examples

Code	Before FS0	FS1	A0	After NCZV	A0
`ZEXT A0,0`	32	x	>FFFF FFFF	xx0x	>FFFF FFFF
`ZEXT A0,0`	31	x	>FFFF FFFF	xx0x	>7FFF FFFF
`ZEXT A0,0`	1	x	>FFFF FFFF	xx0x	>0000 0001
`ZEXT A0,0`	16	x	>FFFF 0000	xx1x	>0000 0000
`ZEXT A0,1`	x	16	>FFFF 0000	xx1x	>0000 0000